

MEMPOWER: A Simple Memory Power Analysis Tool Set

Freeman Rawson
IBM Austin Research Laboratory

January 21, 2004

Contents

1	Introduction	1
2	Different Types of Memory Power Calculators	3
2.1	Simple Calculation from DIMM-level estimates	3
2.2	Spreadsheets Models	3
2.3	Trace-based Energy-per-operation Calculations	4
2.4	Trace-based Time-and-utilization Calculations	4
2.5	Trace-driven Simulation	4
2.6	Execution-driven Simulation	5
3	MEMPOWER Theory and Structure	6
3.1	Inputs	7
3.1.1	Specification Files	7
3.1.2	The Mambo-generated Trace Format	10
3.2	PID Splitting	11
3.3	Trace Reformatting	11
3.4	Spatial Splitting	12
3.5	Interarrival-Time Calculation	14
3.6	Diagramming	14
3.7	Utilization Calculation	16
3.8	Power, Energy and Delay Calculations	17
3.8.1	DDR SDRAM Parts	18
3.8.2	Support-Chip Power Models	24

3.8.3	Summarization Calculations	25
3.8.4	Implementation	26
3.9	Outputs	26
3.10	Post-processing	27
4	Validation and Performance	28
4.1	Validation	28
4.2	Implementation Considerations	29
4.3	MEMPOWER's Performance	29
5	A Sample MEMPOWER Analysis	31
5.1	Benchmark	31
5.2	Memory Layout and Parameters	32
5.3	Trace and Power-Management Characteristics	32
5.4	Interarrival Times	32
5.5	Whole Memory Results	32
5.6	Results for Individual DIMM Groups	33
5.7	Per-Process Power Consumption	33
6	Future Work	37

Abstract

MEMPOWER is a tool set for analyzing memory traces to determine utilization and calculate the power and energy consumption of the memory hardware. The goal of MEMPOWER is to provide a simple, relatively quick-to-run estimation of memory power consumption based on how a workload uses the memory and the technologies used to implement the memory subsystem. It also allows one to experiment with different power-management policies and parameters. In addition to calculating memory power and energy, MEMPOWER also computes the performance impact of power management in terms of the number of cycles of delay injected by power-management actions.

The major contribution of MEMPOWER is to provide an intermediate step in the power analysis of memory subsystems between simple spreadsheet models such as the one provided by Micron [1] and full, power-aware simulations of memory. Recent uses of it include the evaluation of cooperative, hardware-software, power management of memory as reported in a recent paper [2].

This report describes the theory, structure and implementation characteristics of MEMPOWER. It also provides a sample evaluation of memory power and energy under various policies based on a memory trace taken with the Mambo full-system simulator [3].

Chapter 1

Introduction

MEMPOWER is a tool set for analyzing memory traces to determine utilization and calculate the power consumption of the memory hardware. The goal of MEMPOWER is to provide a simple, relatively quick-to-run estimation of the power and energy consumption of system memory based on how a workload uses it and its implementation.

MEMPOWER's features include

- utilization, power and energy data broken down by physical layout of the memory
- support for different types and manufacturers of memory components and different memory timings
- division of the input trace to match the addressing of different physical portions or subdivisions of memory
- division of the input trace by process identifier to allow for the calculation of memory utilization, power and energy values for individual processes
- calculation of request interarrival times by physical unit
- determination of the effects of various power-management policies such as power-down and self-refresh on memory performance, power and energy.

Its results provide not only power and energy information but also insights into how the traced workload uses the different parts of physical memory. It has the additional advantage that one can do a full analysis of a substantial memory trace of a server executing a standard benchmark in a few days using a commodity, multiprocessor workstation. Recently, MEMPOWER was used to do all of the evaluation of the effects of various power-management policies on the behavior of the memory for a paper on cooperative memory power management [2].

This report describes

- the relationship between MEMPOWER and other memory power calculators
- the inputs that it requires including the trace and information about the memory system
- the theory behind it and how it calculates its outputs
- the outputs that it produces
- its validation and performance

- a sample evaluation of a TPC-W trace done using it
- its strengths and limitations
- possible future enhancements.

The major contribution of MEMPOWER is to provide an intermediate step in the power analysis of memory subsystems between simple spreadsheet models such as the one provided by Micron [1] and full, power-aware simulations of memory. Current work in the IBM Austin Research Laboratory is developing a full, power-aware simulation of memory, known as MEMSIM [4]. The intent over the longer term is to integrate this simulation into the Mambo full-system simulator [3], thus providing a much more accurate and fully integrated memory simulation in the Mambo environment. When the fully integrated memory simulator is complete, it will supersede the work reported here.

Chapter 2

Different Types of Memory Power Calculators

In order to understand the motivation for MEMPOWER, it is essential to put it into context, and the best way to do so is to use a taxonomy of the possible ways of analyzing and modeling memory power. There are six (6) possible types of memory power models as described in the following sections.

2.1 Simple Calculation from DIMM-level estimates

In this type of calculation, one does a simple multiplication of the number of DIMMs in the machine times the power per DIMM as quoted by the vendor. Although amazingly simple, this approach is often used in practice. It has the advantage of simplicity, but it is prone to inaccuracy. Since vendors generally overstate DIMM-level power requirements for legal and business reasons, the estimates that it yields are generally higher than either the maximum possible memory power or the largest values measured in practice. However, this is by no means guaranteed.

2.2 Spreadsheet Models

The second type of memory-power-calculation methodology uses a spreadsheet to calculate the power based on currents, voltages, and simple usage models. A good example of this scheme is the spreadsheet that is available from Micron. The major limitation of this model is that it becomes very complex to get all of the usage data into it, especially when the usage is derived from a memory trace. However, this method gives more accurate results than the first approach and is also often used by practicing engineers.

2.3 Trace-based Energy-per-operation Calculations

If a trace of the memory references made by a running workload, such as one produced by Mambo, is available, one can perform energy-per-operation calculations to determine the energy consumption of the memory over intervals of time and the average power dissipated during each interval. In an energy-per-operation scheme, the program assigns energy values, E_{read} and E_{write} , to the read and write operations, respectively. It then divides the trace into time intervals. Within each interval, it counts the number of memory operations – reads and writes – yielding the read count, N_{read} , and the write count, N_{write} , for the interval. If the length of the interval in time is T , then the total energy over the interval is $N_{read} \times E_{read} + N_{write} \times E_{write}$, and the average power over the interval is $(N_{read} \times E_{read} + N_{write} \times E_{write}) \div T$.

2.4 Trace-based Time-and-utilization Calculations

MEMPOWER is an example of a trace-based, time-and-utilization power calculator. Such a power calculator consumes memory traces with timing information in them and produces average power values for one or more intervals of time based on that information, memory service-time parameters, if such information is not present in the trace, and vendor-provided voltages, currents and part counts. From the average power over the interval or intervals, it can then calculate the energy. The equations used in the power calculation are adapted from those published in technical notes from the vendors. If power-management policies are also considered, such calculators can compute delay metrics for different power-management policies and parameters.

MEMPOWER attempts to do a relatively complete calculation and takes into account the power contributions from the support chips required on the DIMM as well as the SDRAM parts. It also incorporates support for all of the major power-states supported for DDR SDRAM chips. Finally, it allows one to calculate the power over the whole trace or the power and energy over a set of time-based intervals or a set of intervals representing the execution of different processes on the machine.

2.5 Trace-driven Simulation

Another way to calculate memory power is to use a trace-driven simulation of the memory subsystem. The simulation tracks the activity of the various components of the memory subsystem and simulates the currents drawn. Based on the currents and known voltages, it can then dynamically calculate the power consumption of the memory over intervals of different sizes. As the size of the intervals used approaches 0, it calculates something approaching the instantaneous power dissipation of the memory. The MEMSIM simulator [4] is an example of a trace-driven simulation of memory activity and power. Since the level of detail is much higher, trace-driven simulators like MEMSIM give more accurate power numbers and much more detail on the performance effects of memory power-management than time-and-utilization-based calculators like MEMPOWER can.

2.6 Execution-driven Simulation

Execution-driven simulation is similar to trace-driven simulation in that it simulates the behavior of the memory subsystem and calculates power based on the activity of the memory. However, the simulation framework and the source of the memory requests are different. One particularly interesting variant of execution-driven simulation is to use a full-system simulator such as Mambo to drive the memory simulation, effectively incorporating it into Mambo. In fact, the current direction of the MEMSIM work is to do just that. Execution-driven simulation is the most complex way to implement a power calculator for memory, but it yields the most accurate results.

Chapter 3

MEMPOWER Theory and Structure

As a trace-based time-and-utilization power calculator, traces of memory references including operation type, physical address and timing drive MEMPOWER. Although there are actually two versions of MEMPOWER, the one described here consumes memory traces generated by the Mambo full-system simulator. The other version processes traces that are in a format used internally by IBM in gathering traces directly from the hardware. That version offers only a subset of the features described here. MEMPOWER calculates memory power and energy for memory sub-systems that are typical of IBM's server products. The memory parts have a number of characteristics.

- DDR SDRAM technology
- registered memory
- ECC
- closed-page operation
- multiple DIMM and/or multiple controller and multiple DIMM subsystems

The details of the speed, currents and layout of the memory subsystem are parameters to MEMPOWER. Calculating power and energy for other types of memory such as DDR-2 or RAMBUS requires minor changes to the tools.

The memory power calculation computes the average memory power over an interval for a defined portion of memory using a time-and-utilization-based methodology. It does its energy calculations by performing an approximate integration of the power over the time intervals. MEMPOWER calculates the delay introduced by the use of memory power-management by tracking the total number of cycles spent recovering from the power-management actions taken during an interval. Doing these computations involves the following general steps.

- **Specification:** specifying the parameters including the details of the memory subsystem and its timings.
- **Process splitting:** optionally splitting the original trace into multiple traces, each of which contains the memory references associated with a single process or other software-defined entity. MEMPOWER does this step only if the analysis is of per-process power and the potential differences in memory-power usage by different processes.
- **Reformatting:** converting the trace into a format that is more conveniently processed by the remainder of MEMPOWER.

- **Spatial splitting:** splitting the trace by address to match the layout of the physical memory. Depending upon the configuration information, this step may be trivial.
- **Interarrival calculation:** adding interarrival time information.
- **Collection of interarrival-time data:** optionally collecting information about interarrival-time statistics.
- **Diagramming and delay introduction:** converting the trace to a format which tracks the state of the memory over time, using the memory-service times specified as parameters. This step includes introducing the delays associated with any power-management policy being applied to the trace.
- **Utilization calculation:** calculating the utilization of each part of the memory over a set of intervals or the whole trace.
- **Power calculation:** calculating the average power over each interval or the whole trace. This step also calculates the total energy for the interval or trace as well as any delay injected by the use of power-management modes.
- **Post-processing:** converting the power, energy and delay numbers into human-readable output and graphs.

MEMPOWER optionally splits a trace by process and typically splits it spatially by address. The subsequent processing steps apply to the results of these splits, so that if there are, for example, 100 processes and sixteen (16) memory subunits, it runs the remaining steps 100×16 or 1600 times. Most evaluations do not require a per-process split and have far fewer subtraces to handle. Figure 3.1 illustrates the main steps in processing a trace with MEMPOWER while Figure 3.2 shows a secondary set of steps used in interarrival-time processing.

Since MEMPOWER is capable of applying a number of different hardware-based, memory-power-management policies with different parameters to the same trace, one need collect only a single trace with no hardware-based power management and run it through MEMPOWER many times with different power-management policies and parameters to determine the effects of using the policies. Since MEMPOWER processing is generally significantly faster than trace collection, this reduces the time required to explore the space of memory-power-management options.

Rather than being a single, very large, integrated program, MEMPOWER is a set of tools that one runs in sequence to do the desired memory power, energy and delay calculations. As a side-effect, MEMPOWER also generates a set of memory utilization statistics since, as indicated previously, MEMPOWER uses a time-and-utilization-based model for memory power.

The following sections describe MEMPOWER's inputs, theory, processing steps and outputs.

3.1 Inputs

MEMPOWER needs two distinct types of input – specification files and a trace of memory references.

3.1.1 Specification Files

Whenever possible, MEMPOWER is parametric to maximize its flexibility. To operate, it needs a description of the memory subsystem including how ranges of addresses map to parts, the currents drawn by the various memory parts when in different states, read and write service-time values and information about the power-management policies,

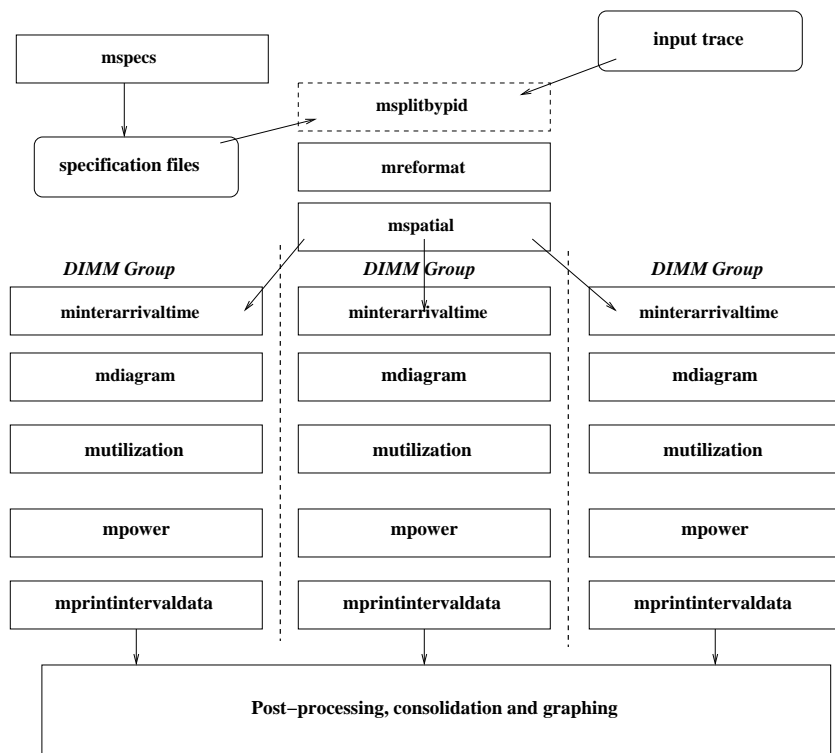


Figure 3.1: MEMPOWER processing steps

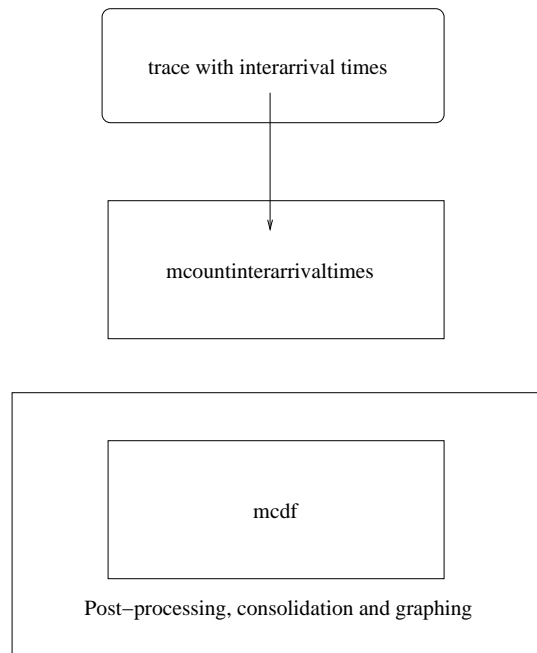


Figure 3.2: MEMPOWER processing steps for counting and processing interarrival times

if any, that MEMPOWER is to apply to the input trace. Rather than using command-line parameters and scripts to provide most of this information, MEMPOWER processes *specification files*. There are five types of specification files.

- **Memory:** describes the characteristics of the DRAM parts used including the currents drawn in the various operating states, the number of pins and the voltages.
- **Register:** provides the currents drawn by the register parts used on registered DIMMs and their operating voltage.
- **PLL:** gives currents and voltage for the PLL or timing part used to generate the DIMM-level clock.
- **DIMM:** provides the number and organization of the parts on a single DIMM including the bank structure. This file also includes the read and write service-time values used in the calculations.
- **System:** describes the overall layout of the memory including its total size, the number of different, separately manageable portions of memory and the interleaving scheme, if any. It also gives certain features of the system such as the processor speed and the cache-line size.

Each file is a set of lines of the form $\langle \textit{keyword} \rangle = \langle \textit{value} \rangle$. Rather than having each program in MEMPOWER parse text files, there is single, separate program, `mspecs`, that converts all of the specification files for a particular configuration to a set of binary files. Each MEMPOWER program reads the binary form of the files that are relevant to it directly into internal data structures.

The only exception to this general strategy is the specification of the power-management policies. Since MEMPOWER needs different delay and threshold values for each power-management policy that it tries and since only a single program, `mdiagram`, needs them, they are command-line parameters. This makes it easy to write a wrapper script that explores the implications of many different power-management options and avoids the creation of a very large number of specification files.

3.1.2 The Mambo-generated Trace Format

As a trace-driven power calculator, MEMPOWER also requires a trace of memory references to process. Although these traces may come from different sources and although there is a version of MEMPOWER that processes traces collected directly from hardware, the version described in this report uses memory-reference traces collected using emitter logic in the Mambo [3] full-system simulator. The trace format is a simple one that consists of a sequence of records with most of the records describing references to the (simulated) real memory of the simulated system. These records contain the following fields.

- `type`
- `data`
- `dt`
- `alist`

The `type` indicates whether the record describes a memory read or write or some other kind of information. The `data` field is, for memory operations, the physical address referenced while the `dt` field is the number of processor clocks since the previous record. The `alist` field is specific to the particular implementation and is used in the evaluation in Chapter 5 to indicate which portions of memory the software has turned off, independently of any hardware-based power management. Section 3.4 describes how MEMPOWER divides memory into separately power-manageable units. Since there is no length information, MEMPOWER assumes that all memory operations are for the number of bytes in a single cache line.

Other than memory reference operations, the traces contain information about process creation, switching and termination. From this information, MEMPOWER can determine the last process to gain control and, thus, what process should be charged for the subsequent memory operations.

Each trace file has three distinct *length* attributes associated with it. The first, and least important, length attribute is the number of bytes in the file. The primary effect that this form of length has on MEMPOWER is that to conserve disk space, one may need to compress the stored trace files, increasing the overall running time of a MEMPOWER analysis. The second length attribute is the number of trace records in the file. This length attribute affects the running time of the MEMPOWER steps by determining the number of I/O operations the programs must do. Finally, the third length attribute, referred to here as *temporal length*, is the length of the trace in processor clocks; that is, it is the sum of all of the `dt` values in the trace.

3.2 PID Splitting

MEMPOWER allows one to calculate the power and energy consumed by individual software-defined entities present in the trace, typically processes as identified by process identifiers or *PIDs*. Rather than complicating its internal representations, MEMPOWER provides an optional, initial processing step, `msplitbypid`, that divides the original trace into multiple traces, all in Mambo format, with one output trace for each process. Thus, if the trace contains references to 100 PIDs, `msplitbypid` produces 100 output traces. Each process trace includes all of the memory references charged to the particular process by MEMPOWER. Each trace is of a subset of the original trace with a length in processor clocks equal to the number of processor clocks that the process has control in the original trace. For a uniprocessor, the sum of the temporal lengths of the subtraces is equal to the temporal length of the original trace while for a multiprocessor, the sum is equal to the temporal length of the original trace times the number of processors. All time, even idle time, is charged to a process. MEMPOWER executes `msplitbypid` only if it is doing per-processor memory power calculations.

There are cases in which some of the processes in the trace, such as, for example, the Mambo kernel thread, generate memory references that are irrelevant to or distort the study being done. To handle such situations, MEMPOWER also has a program, `mfilteroutpid`, that removes all records from the original trace associated with one or more PIDs that are not of interest.

3.3 Trace Reformatting

The first mandatory step in MEMPOWER processing is to convert the Mambo-generated trace to a format that is easier for the remaining programs in MEMPOWER to process. This reformatting, done by `mreformat`,

- converts the relative times to absolute time values, calibrated in processor clocks and with a clock value of 0 for the starting time of the trace
- drops all trace records except those that are memory reads or memory writes
- incorporates the current process id in each record.

The result is a new trace of with approximately as many records as the original that tracks all of the memory activity by address and process id. Since the first memory reference in the trace may have a non-zero relative time, the first record in the reformatted trace may have a non-zero absolute time. The traces currently generated by the Mambo are uniprocessor-only, and, thus, there is no provision for a maintaining a processor identifier in the reformatted trace. PID tracking is done by maintaining the current PID based on the process creation, switch and termination and the program execution events present in the original trace but removed by the reformatting process.

One may think of the step that converts relative to absolute times as *trace clocking*. Subsequent processing steps, in some cases, re-clock the trace to account for delays introduced by power managing the memory.

3.4 Spatial Splitting

For the purposes of memory power calculation, the most important features of the memory subsystem are its division into multiple components that are separately power-manageable and the way in which the memory subsystem maps the physical addresses generated by the processor or processors to the memory components. To manage power, the hardware may group some number of separate components into a single group that it manages as a unit. For example, it may treat a memory controller and all of the DIMMs controlled by it as a single power-management unit. Although it is possible that the individual components within a group are subject to additional power-management actions, MEMPOWER does not attempt to take this into account. Once the power-manageable units are defined, MEMPOWER always treats each one as an indivisible entity. As an example, although they do not implement memory power-management, the IBM Regatta systems [5] have multiple memory controllers that, in theory, are separately manageable for power.

The other important feature of the memory subsystem that MEMPOWER takes into account is how the memory hardware maps the physical addresses generated by the processor or processors to the power-manageable units of memory. Some architectures use interleaving to extract additional parallelism from the memory. Other architectures offer configurable interleaving or do a packed assignment of addresses to physical memory. MEMPOWER's spatial processing breaks up memory traces into sub-traces of the activity against each of the power-manageable units, taking any interleaving of addresses across these groups into account.

MEMPOWER uses the term *DIMM group* to describe spatial units into which it splits memory. Each DIMM group is the same size in terms of the amount of memory and the number of individual parts such as DIMMs and SDRAM chips that it contains. The DIMM-group size is a parameter to MEMPOWER so that one can use it to analyze traces with different memory sizes and different memory layouts. Each DIMM group is thought of as being somewhat independent of all of the others. For example, a DIMM group may be a memory controller plus all of the components it controls, a memory interface plus its associated DIMMs, a group of DIMMs or a single DIMM. Related work such as PAVM [6] and cooperative DRAM power management [2] use terms such as *node* and *memory module* instead of *DIMM group*.

MEMPOWER associates one or more DIMM groups together into an *interleaving group* in such a way that the interleaving groups cover all of physical memory. An interleaving group represents a portion of memory to which the system maps a contiguous range of physical addresses. The system interleaves consecutive cache lines across the members of a single interleaving group before proceeding to the next interleaving group. Intuitively, the concept of an interleaving group arises from the ability of some systems such as those in IBM's pSeries family to interleave successive cache lines across multiple memory controllers. MEMPOWER requires that the total memory size, the number of interleaving groups and the number of DIMM groups be such that all interleaving groups have the same number of DIMM groups in them. Figure 3.3 shows a very simple interleaving and DIMM group configuration.

MEMPOWER assumes that all memory references are cache-line reads and writes, and the size of a cache line is a parameter to it. This is a reasonable assumption since, in practice, essentially all of the memory traffic found in the input traces is in cache-line-sized units and aligned on cache-line boundaries.¹ If i is the number of interleaving

¹The Mambo trace format also assumes this and does not bother to encode the length of the request in the trace records that it produces.

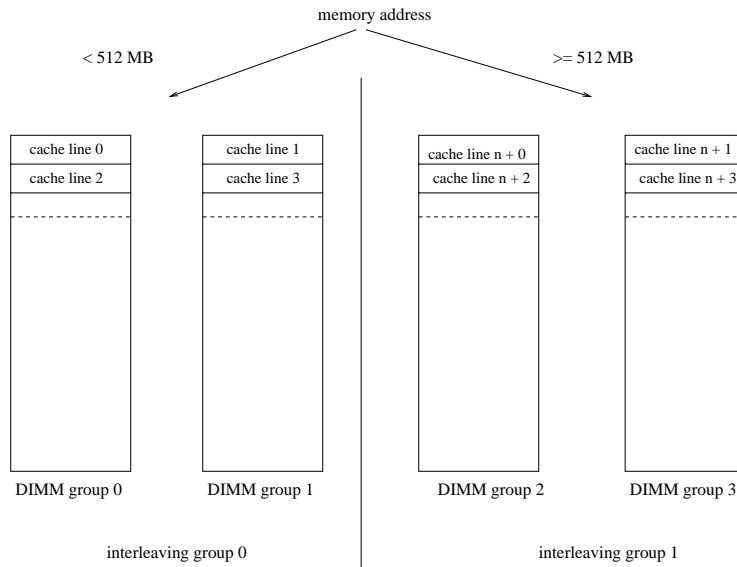


Figure 3.3: Sample interleaving and DIMM group structure with two (2) interleaving groups, four (4) DIMM groups and a total memory size of 1 GB. The bytes of a cache line are assigned to a DIMM group based on the address of the first byte.

groups, n is the total number of DIMM groups, m is the total memory size and l is the cache-line size, then given an address a , its DIMM group d is given by the following equation.

$$d = (\lfloor \frac{a}{\frac{m}{n} \times l} \rfloor) + remainder(\lfloor \frac{a}{l} \rfloor, i)$$

In most cases, the interleaving group is the natural unit of power management since the system maps consecutive physical addresses to it, and MEMPOWER assumes that the system puts all of the parts in an interleaving group into the same power state. (In the example of using MEMPOWER presented in Chapter 5, each interleaving group contains precisely one DIMM group, and the concept of interleaving group is not used in describing such configurations.)

Thus, the next step in MEMPOWER processing, implemented by the `mspatial` program, divides the trace into multiple subtraces by address range, with each subtrace representing a single DIMM group. The `mspatial` program accepts a single file in the format written by `mreformat` and produces multiple files in the same format. The output files are either empty, if there are no references to the DIMM group in the input trace, or are of approximately the same temporal length as the input trace. The intuition behind `mspatial` is that MEMPOWER is generating a trace of the activity of each subcomponent of memory. As described above, MEMPOWER's primary application is in the calculation of memory power for memory subsystems with multiple interleaving and DIMM groups. The spatial split done by `mspatial` takes parameters that describe the memory and cache-line sizes, the number of interleaving groups and the number of DIMM groups. The `mspatial` step is omitted if MEMPOWER is calculating the power consumption of the whole memory rather than its individual parts. Once the original trace is spatially split,

Scans of the trace format used by the other version of MEMPOWER indicate that there is no traffic to and from memory other cache-line accesses.

all subsequent processing is done on a per-DIMM-group basis.

3.5 Interarrival-Time Calculation

The next step of the MEMPOWER calculation, implemented by the `minterarrivaltime` program, converts each of the non-empty traces resulting from the spatial splitting to a different format that adds additional information to each record in the trace about the time since the previous memory operation and the previous memory operation of the same type (read or write). MEMPOWER treats empty DIMM-group-level traces differently, and the next step in MEMPOWER processing handles them appropriately. In cases where the trace records interarrival times of 0, MEMPOWER makes no attempt to serialize them by moving all arrivals with time 0 after the initial one in such a sequence later in time. The output traces from `minterarrivaltime` are the input into the next step of MEMPOWER. An experimental form of a program to serialize the DIMM-group traces exists, but so far it is not clear that using it yields a more accurate estimate of the power. No systematic memory-power analysis done to date has used it.

In addition to `minterarrivaltime`, there is a second program in MEMPOWER, called `mcountinterarrivaltimes` that counts the number of memory references with each distinct interarrival time value. This allows a post-processing program to create a random variable for the interarrival-time values for a single DIMM group. The `mcountinterarrivaltimes` program takes as its input the output traces from `minterarrivaltime` for each DIMM group and produces a unique output file that feeds directly to a post-processing program as shown in Figure 3.2.

3.6 Diagramming

Once MEMPOWER calculates the interarrival times, the next step is to convert the trace to what is called a *diagram* format by running the `mdiagram` program on it. The result is a set of records describing the entire time covered by the trace, explicitly including the idle periods and any time during which the memory is recovering from a power-management action. A very loose analogy with the timing diagrams used in electrical engineering motivates the terminology. The primary advantage of the diagram format is that it makes it easy to calculate the fraction of time that the memory is in each state. The `mdiagram` program reads the output from `minterarrivaltime` and creates the diagram for a DIMM group.

As Mambo does not simulate the details of the memory, there is no information in the input trace about the time that each memory operation takes. Instead, the MEMPOWER user must estimate the read and write service times in processor clocks and pass them as parameters to the `mdiagram` program using the DIMM specification file. As discussed in Chapter 4, this estimate of memory service times is one of the major potential sources of inaccuracy in a MEMPOWER evaluation. The `mdiagram` program handles memory references of the same type that occur during the service of a prior memory reference by extending the length of the current (active) state of the memory. Memory references of the opposite type cause a state transition immediately after the current reference or string of references of the same type finishes.

Since the goal of MEMPOWER is to allow one to compare different memory-power-management policies, one must allow for different power-management policies with various delay and threshold combinations. MEMPOWER supports three policy types – power-down, self-refresh and a combination policy of power-down followed by self-refresh after an additional period of idleness. To incorporate the effects of power-management policies, `mdiagram` has features that allow one to apply a particular power-management policy to a trace and get a diagram that is specific to it. The `mdiagram` program accepts command-line parameters giving the delay values and idleness thresholds for power-down and self-refresh. Using the default values of 0 for all of these parameters gives the base, unmanaged diagram described above. Giving a non-zero value for the power-down delay and a 0 value for the self-refresh delay yields a power-down policy. Similarly, giving a non-zero value for the self-refresh delay and a 0 value for the power-down delay gives a self-refresh policy while using non-zero values in both delays causes MEMPOWER to apply the combination policy. MEMPOWER uses inelastic or non-absorbing delays to account for the resynchronization of memory following a power-management action. This means that all subsequent operations in the trace are delayed by the accumulated delay injected by power management to the point in the trace where they occur. In other words, if the operation occurs at time t in the original trace and there is an accumulated delay d_t due to power management at time t , the operation actually occurs at time $t + d_t$. If at the beginning of an interval, the accumulated delay is d and power management during the interval adds e more cycles of delay, the injected delay for the interval is e , and the accumulated delay at the end of the interval is $d + e$.

The resulting diagram is a set of records that give an initial time, a duration and an operating state for each different state that the memory enters during the trace. The durations from the diagram records sum to the temporal length of the trace. The operating states are

- idle stand-by
- read
- write
- power-down
- self-refresh
- recovering before a read operation
- recovering before a write operation.

If there are no references to a DIMM group in the original trace and `mdiagram` is not applying any power-management policy, it generates a diagram with a single record that covers the entire temporal length of the trace in an idle state. It uses an input parameter to provide the required length in processor clocks. If `mdiagram` is applying a power-management policy, it uses the threshold value or values specified to generate one or two subsequent records covering the low-power state or states of the memory from the threshold point or points through the temporal length of the trace.

Figure 3.4 is a graphical depiction of a small segment of a diagram file.

In addition to the standard policies, MEMPOWER also has an experimental hybrid power-management policy. This policy is a combination power-down, followed by self-refresh policy that uses the PID of the current process to select the threshold for going to self-refresh mode. The underlying idea is that different processes have different levels of memory activity for different DIMM groups. Processes that actively use a particular DIMM group may save energy by having a higher idleness threshold before transitioning from power-down to self-refresh due to the relatively long

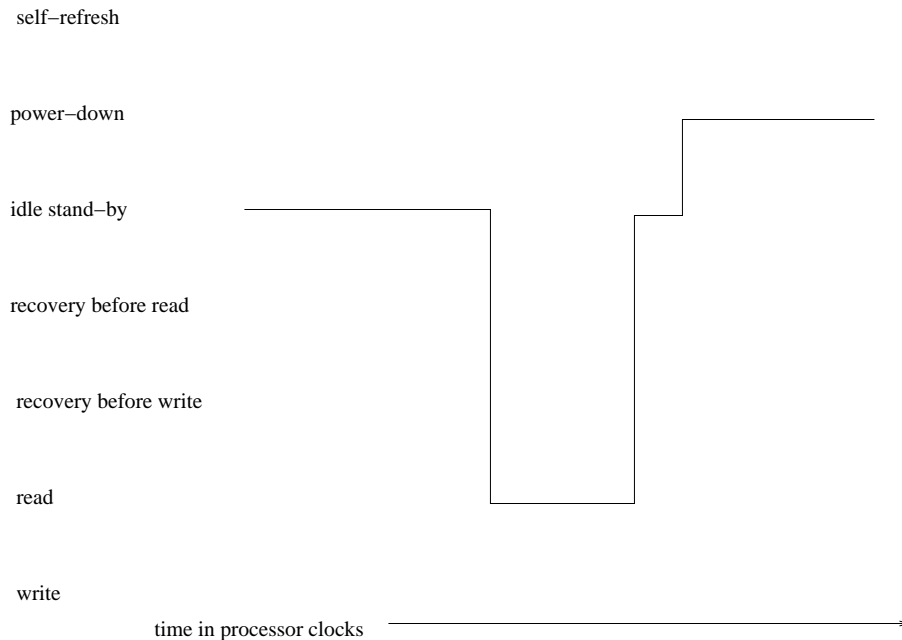


Figure 3.4: A conceptual representation of a small segment of a diagram file

latencies associated with recovering from self-refresh mode.

Since each run of `mdiagram` produces a diagram file that describes the memory states for the DIMM group for a particular combination of power-management policy and parameter values, one must run `mdiagram` and the programs following it in the MEMPOWER chain repeatedly, once for each distinct policy that one wishes to consider.

3.7 Utilization Calculation

Using a diagram as input, the next step in the MEMPOWER calculation, implemented by the `mutilization` program, computes the fraction of time spent in each state. This calculation may produce a single value for the whole diagram or a set of values for equal-length time intervals. Typical interval lengths are on the order of 100 milliseconds although there is no real restriction on the size or number of intervals.

MEMPOWER reads a diagram file and creates a binary file that contains the utilization and power data for the DIMM group. There is one record in this file for each interval. Depending on how the user runs MEMPOWER, the file may contain one record representing the utilization for the DIMM group over the entire duration of the trace, or it may contain multiple records, one for each interval of the trace. The output file is called an *interval-data file* and is the input to the power calculation step of MEMPOWER. In each interval, MEMPOWER counts

- the length of the interval in clock cycles
- the number of memory read operations

Description	Symbol
Number of cycles in the interval	n_{total}
Count of read operations	c_{read}
Number of read cycles	n_{read}
Count of write operations	c_{write}
Number of write cycles	n_{write}
Number of cycles in power-down	n_{pd}
Number of cycles in self-refresh	n_{sf}

Table 3.1: Operation and cycle count notation

- the number of clock cycles that the memory spends reading
- the number of memory write operations
- the number of clock cycles that the memory spends writing
- the total number of clock cycles that the memory is active
- the number of clock cycles that the memory is in idle stand-by
- the number of clock cycles that the memory is in power-down
- the number of clock cycles that the memory is in self-refresh
- the total number of clock cycles of injected delay due to the re-synchronization needed after the application of a power-management policy
- the average injected delay per memory operation.

MEMPOWER calibrates all of the values except the counts of read and write operations in processor clocks, and one can translate these values into ordinary time values using the known frequency of the simulated processor. At this point, MEMPOWER treats all elements of a DIMM group as being in the same operating state at the same time. Table 3.1 summarizes the notation used for some of these values in Section 3.8.

3.8 Power, Energy and Delay Calculations

MEMPOWER calculates the memory power and energy for the whole DIMM-group trace or for each interval using the counts and times computed in the previous step. These calculations use the equations and assumptions given in this section as well as the voltages and currents given in the manufacturers' datasheets and encoded in the specification files. MEMPOWER supports optional scaling of the vendor's current values based on experience.

The power calculation consists of two separate calculations, one for the power of a single SDRAM part under the presented load and one for per-DIMM overhead of the memory packaging. MEMPOWER combines the results of these computations together to produce a final power number for an interval. The power values calculated depend on

- the organization of the memory
- the memory access pattern

- the use or non-use of various power-saving modes available from the memory components
- current and voltage values for the parts used
- the number of memory components involved.

The power metric calculated by MEMPOWER is the average power dissipation over an interval of time of the memory parts. For short intervals of time, which are not necessarily always used in MEMPOWER calculations, the average power becomes approximately equal to the instantaneous power dissipation at any time in the interval. This, of course, begs the question of the actual power dissipation as a continuous function of time, but due to the nature of the workloads of interest, which exhibit fairly constant load at a micro time-scale and the limitations on the response of physical devices, it is quite likely that the variance in the power dissipation is small over short intervals of time, and the curve is relatively flat. Even so, although it is beyond the scope of the current work, this question does bear further investigation.

Once it calculates the average power over an interval, MEMPOWER then computes the energy used over the interval by multiplying the average power by the length of the interval in standard time units such as milliseconds. The total energy consumption over the whole trace is best calculated by calculating the power and energy over relatively short intervals and adding. In effect, MEMPOWER does an approximate integration to get the energy, and its accuracy is dependent on having intervals that are short enough that the power in each interval accurately represents an instantaneous value. Also the power calculation passes through the values for the total delay and the average delay per operation injected in each interval by whatever power-management policy, if any, is being applied that were computed during the utilization step. These delay values allow one to quantify the performance impact of power management on the memory.

The following subsections describe in more detail the theory behind the power calculation and the operation of the `mpower` program that implements it.

3.8.1 DDR SDRAM Parts

Most of the parts on a DIMM are DDR SDRAM chips, and the basis for modeling the power that they consume is a power modeling methodology for individual SDRAM parts described by Micron in an application note on DDR SDRAM power calculations [1]. MEMPOWER does not attempt to deal with the additional currents and states of DDR-2 memory, and it also has no logic for calculating the power of a RAMBUS [7] memory. However, the extensions to cover these cases should be relatively simple.

To get the average power consumption of an SDRAM part over an interval, the state of the SDRAM is first categorized into distinct phases. Each phase corresponds to a certain type of activity for which the data sheet for the part contains the average measured current. The power, P , consumed during that phase can then be computed as the product of the data sheet (or measured) current, I , for that phase and the corresponding voltage, V , driving the current. Equation 3.1 summarizes this calculation.

$$P = I \times V \tag{3.1}$$

During each interval considered and for every memory access pattern of interest, one derives the fraction of time the SDRAM is in each phase. Knowing the fractions, the average power consumption is then the weighted sum of the power consumption for each phase. For example, if the memory access pattern over the interval has just two phases

Description	Symbol
<i>read</i>	I_{dd4R}
<i>write</i>	I_{dd4W}
<i>active stand-by</i>	I_{dd3N}
<i>pre-charge stand-by</i>	I_{dd2F}
<i>active power-down</i>	I_{dd3P}
<i>per-charge power-down</i>	I_{dd2P}
<i>activation</i>	I_{dd0}
<i>auto-refresh</i>	I_{dd5A}
<i>self-refresh</i>	I_{dd6}

Table 3.2: The operating states of an SDRAM part and the currents associated with them. This table includes the currents for activation, auto-refresh and self-refresh that are used in the equations in this chapter.

with corresponding usage fractions of f_1 and f_2 , currents I_1 and I_2 , and voltages V_1 and V_2 , then the average power consumption is given by Equation 3.2.

$$P = f_1 * I_1 * V_1 + f_2 * I_2 * V_2 \quad (3.2)$$

Normally, and in all cases considered here, the voltages are identical in all operating states and are taken to be equal to the nominal maximum voltage, V_{dd} and later adjusted to the operational supply voltage of the chip. The use of the nominal maximum voltage is essential since the currents specified by the vendors are measured at that voltage.

In MEMPOWER's modeling of the part, a DDR SDRAM has many possible states distinguished by the responsiveness of the device when inactive and the nature of activity when active. The models presented here identify seven different phases in the operation of the SDRAM part:

- *read*
- *write*
- *active stand-by*
- *pre-charge stand-by*
- *active power-down*
- *pre-charge power-down*
- *self-refresh*.

Table 3.2 gives the symbols used to refer to the currents associated with these states.

Not all of these phases are not necessarily used. Rather their usage depends on the access pattern and the power-management policies applied to the SDRAM. Each phase has a distinct current value associated with it, and Table 3.2 gives the labels that the datasheets with them.

During normal operation the device receives a clock (CLK) input and a clock enable (CKE) input that indicates a valid CLK signal. A fast-response, inactive, low-power mode for the device is *power-down* which the SDRAM enters

when the CKE input is false. Power-down typically lowers the power consumption of an idle device by 80% or more and requires a delay of 1 memory cycle or 2 data-transfer cycles for re-activation. For stable exit from *power-down* the hardware must maintain the signal frequency at the CLK input as during normal operation. Further, it must issue periodic refresh commands to maintain the data content in memory.

It is possible to save additional power by entering the *self-refresh* mode. In this mode, both the CLK input and periodic refresh commands are unnecessary. Instead, the device uses an internal refresh mechanism that maintains the integrity of the data stored in the device. The additional power savings are obtained by placing the clock-generation circuitry in its power-down mode and by dispensing with the external refresh generation process. However, the power consumption of the SDRAM device itself is very close to that at power-down. Moreover, the delay to come out of *self-refresh* is significant, needing about 200 memory cycles or 400 DDR transfer-slots for re-synchronization with the external CLK, upon re-activation. As a consequence of this delay, there is a trade-off between the added power benefits and the performance penalty of using *self-refresh*. One of the objectives of MEMPOWER is to make it possible to study these trade-offs as well to investigate combination policies that enter *power-down* after an initial period of inactivity and then go to *self-refresh* after an additional period of idleness.

When not in *self-refresh* and independently of the CKE signal, that is, whether the device is in *power-down* or not, the device can be in one of two states – *pre-charge* or *active*. SDRAM devices store data in cells that are organized by rows and columns. Data are read and written to the memory cells in the device through intermediate buffers (or sense amplifiers) that accommodate the data for an entire row of SDRAM cells: data are read or written to SDRAM cells at a row-granularity through the buffers. When the data corresponding to a row of cells are ready in the buffer to be read or modified, the device is said to be in the *active* state. Otherwise, it is said to be in the *pre-charge* state. These states have different implications for the device's responsiveness and power consumption. These two states combined with the CKE status yield four of the six distinct phases of operation identified previously – *active standby*, *pre-charge standby*, *active power-down* and *pre-charge power-down*.

Prior to the voltage adjustment, the power consumption for each phase is just the corresponding current multiplied by the chip supply voltage, V_{dd} . MEMPOWER's power calculations always use only the pre-charge modes, pre-charge standby and pre-charge power-down, for the inactive phases and the active standby mode for the active phases. That is, power-down is triggered only when all the banks on a device are pre-charged. This is a natural fit for closed-page-mode operations where an active bank is automatically pre-charged once the current access to it ends. For open-page-mode operations, where the accessed row is left in the row buffer in active state, the active power-down phase is entered upon power-down. If the currents corresponding to the two power-down phases I_{dd2P} and I_{dd3P} are quite different, then power estimates using only pre-charge power-down phases have some error for systems using open-page mode.

For closed-page mode, both of the non-idle phases, *read* and *write*, have a common power component, that of bringing the SDRAM into the *active* mode. This involves reading the addressed row of SDRAM cells into the row buffer so that the SDRAM can read or modify data. Once the activity is over, the SDRAM goes to one of the pre-charge modes. The non-idle phases break down into three kinds of activity – one for the activation/de-activation pair, one for read, and one for write. For the entire period that the device is active, during activation/de-activation, read or write, it is consuming power corresponding to the active standby phase in addition to the power estimated below for activation and read or write activity. As shown in Table 3.2, the data sheet current corresponding to the activation/de-activation

pair is I_{dd0} . It is the average current drawn when doing the activation/de-activation sequence at a period of t_{RC} nanoseconds. The value of t_{RC} comes from the manufacturer’s datasheet for the DDR SDRAM part. A higher (lower) rate yields a correspondingly higher (lower) current. Further, this average current measurement for the data sheet is performed with the SDRAM in the active standby state, drawing an average current of I_{dd3N} . Taking both into account, if the observed average device activation period for a specific memory access pattern is $t_{RC}(observed)$, the power component corresponding to the activation/de-activation pair is given by Equation eqn:activation-power.

$$P_{activation} = (I_{dd0} - I_{dd3N}) * V_{dd} * t_{RC}/t_{RC}(observed) \quad (3.3)$$

The values for t_{RC} and $t_{RC}(observed)$ are calibrated in nanoseconds.

Prior to calculating $t_{RC}/t_{RC}(observed)$ or any of the fractions used in the final power calculation, MEMPOWER adjusts the operation counts and cycle values for any multi-banking or ranking of the SDRAM chips. It divides c_{read} , c_{write} , n_{read} and n_{write} by a rank count given in the DIMM specification file.

MEMPOWER calculates the value of $t_{RC}(observed)$ by calculating the number of active periods based on the number of individual read, c_{read} , and write operations, c_{write} , initiated in each interval. The sum of the number of reads and writes yields the total number of active periods in the interval. MEMPOWER computes the length of the interval in nanoseconds and divides by the number of active periods to get the value of $t_{RC}(observed)$. This is shown in Equation 3.4 where t_l is the length of the interval in nanoseconds and is given by $t_l = (n_{total}/f) \times 10^9$ if f is the processor clock frequency.

$$t_{RC}(observed) = \frac{t_l}{c_{read} + c_{write}} \quad (3.4)$$

This calculation, of course, results in an averaged value for $t_{RC}(observed)$ that assumes that the activations are uniformly distributed over the interval, and, thus, it is a potential source of error although the error is likely to be small if the intervals are sufficiently short. MEMPOWER interprets the $t_{RC}/t_{RC}(observed)$ ratio as the fraction of the interval that the memory is active. In addition, there are two potential hazards in this calculation that MEMPOWER must handle. First, if the DIMM group is totally idle during the interval or if the only activity is an operation that started in the previous interval, the operation counts sum to 0, and the division needed to get $t_{RC}(observed)$ is illegal. MEMPOWER rectifies this problem by arbitrarily setting the number of active periods to 1 in this case. Second, it is possible that the activation count is high that $t_{RC}/t_{RC}(observed)$ is greater than 1. The source of this problem is the inherent guesswork about memory service times that MEMPOWER uses and the mismatch between the values supplied to MEMPOWER and the behavior of Mambo as it issues and times memory operations. This problem is especially acute since Mambo does not currently model the temporal behavior of memory. To solve this problem, MEMPOWER *clamps* $t_{RC}(observed)$ to 1. That is, if it is greater than 1, it is set arbitrarily to 1.

Power consumption for a write phase is computed as indicated in Equation 3.5

$$P_{write} = (I_{dd4W} - I_{dd3N}) * V_{dd} \quad (3.5)$$

with I_{dd3N} subtracted to account for the active standby power component, which is added separately for any active period. For writes, the power to drive the pins is drawn from the system power supplies and not the DRAM supplies. For a read phase there are two components – the in-device component of

$$(I_{dd4R} - I_{dd3N}) * V_{dd}$$

and an I/O component to drive the data and strobe pins of the device. Interestingly enough, the current per I/O pin, I_{OL} , does not appear to change much from generation to generation and part to part. The I/O pins also have a unique voltage so that their power, P_{perDQ} , is given by

$$P_{perDQ} = (V_{tt} - V_{tt_adj}) * I_{OL}$$

where $V_{tt} - V_{tt_adj}$ is the adjusted termination voltage for each pin. This is multiplied by the number of data pins, nDQ , plus the number of data strobe pins, $nDQS$, for the device to get the total device I/O power consumption by the pins during reads. Thus, the power for a read is given by Equation 3.6.

$$P_{read} = (I_{dd4R} - I_{dd3N}) * V_{dd} + P_{perDQ} * (nDQ + nDQS) \quad (3.6)$$

Every row in an SDRAM device needs to be refreshed once every 64 millisecond at nominal temperatures. When the part is not in self-refresh mode, this implies that the controller must issue an auto-refresh command once every 64 milliseconds divided by the number of rows in the device on average. The power model presented here does not account for the interference of auto-refresh with regular device operation because of the more complex modeling need and the very small perturbation it produces due to its low frequency of occurrence. This decision is a source of error in MEMPOWER's results. However, neglecting the interference probably has a very minor effect compared to that caused by the need to estimate the service times of memory reads and writes. MEMPOWER also simplifies the current expression by considering only I_{dd2P} rather than I_{dd2P} and I_{dd2F} . This too is a source of error. Equation 3.7 gives the auto-refresh power.

$$P_{auto-refresh} = (I_{dd5A} - I_{dd2P}) * V_{dd} \quad (3.7)$$

The power calculations described so far use a V_{dd} value for the device supply voltage. Data sheet current specifications are at the worst-case device operation, that is, at the highest operational temperature and voltage. Devices are actually operated at their nominal voltages. To adjust for nominal voltage operation requires using the fact that CMOS circuit power consumption is proportional to the square of the voltage at a constant frequency of operation. So, with I_{phase} as the current corresponding to a phase, V_{dd} the peak voltage for the part at which datasheet measurements are done, and V_{dd_op} the nominal voltage at which the part is used, the power consumed during the phase is calculated by 3.8.

$$P_{phase} = I_{phase} * V_{dd} * (V_{dd_op}/V_{dd})^2 \quad (3.8)$$

Using the operation and cycle counts generated by `mutilization`, MEMPOWER calculates, for each interval, the fraction of the interval that the memory is in each of the above phases. Table 3.3 defines the notation used below to describe the computation of the required fractions.

Since the I/O pins are active during the *read* phase and not at any other time, $f_{io} = f_{read}$. MEMPOWER uses Equations 3.9 through 3.17 to calculate the fractions needed to compute the average power over the interval. Equation 3.9 is subject to clamping as discussed above.

$$f_{activation} = t_{RC}/t_{RC}(observed) \quad (3.9)$$

Variable	Meaning
$f_{activation}$	Activation current
f_{write}	Write
f_{read}	Read
f_{io}	I/O pins active
$f_{act_standby}$	Active stand-by
f_{act_pd}	Active power-down
$f_{pre_standby}$	Pre-charge stand-by
f_{pre_pd}	Pre-charge power-down
f_{sf}	Self-refresh
n_{total}	Total number of processor clocks in the interval
n_{write}	Write clocks
n_{read}	Read clocks
n_{pd}	Pre-charge power-down clocks
n_{sf}	Self-refresh clocks

Table 3.3: Notation used to describe the phase fractions for an interval

$$f_{write} = n_{write}/n_{total} \quad (3.10)$$

$$f_{read} = n_{read}/n_{total} \quad (3.11)$$

$$f_{act_standby} = (n_{read} + n_{write})/n_{total} \quad (3.12)$$

$$f_{pre_pd} = n_{pd}/n_{total} \quad (3.13)$$

$$f_{act_pd} = 0 \quad (3.14)$$

$$f_{sf} = n_{sf}/n_{total} \quad (3.15)$$

$$f_{pre_standby} = 1 - (n_{read} + n_{write} + n_{pd} + n_{sf})/n_{total} \quad (3.16)$$

$$f_{autorefresh} = f_{pre_standby} + f_{pre_pd} \quad (3.17)$$

Using all of the previous notation, Equation 3.18 gives the average power over an interval at the nominal voltage.

$$\begin{aligned}
P_{nominal} = & (Idd2P * f_{pre_pd} \\
& + Idd3P * f_{act_pd} \\
& + Idd2F * f_{pre_standby} \\
& + Idd3N * f_{act_standby} \\
& + (Idd0 - Idd3N) * f_{activation} \\
& + (Idd4W - Idd3N) * f_{write} \\
& + (Idd4R - Idd3N) * f_{read} \\
& + (Idd5A - Idd2P) * f_{autorefresh} \\
& + Idd6 * f_{sf}) * Vdd \\
& + (Vtt - Vtt_adj) * IOL * (nDQ + nDQS) * f_{io}
\end{aligned} \tag{3.18}$$

Equation 3.1 applies the voltage adjustment.

$$\begin{aligned}
P_{SDRAM} = & (Idd2P * f_{pre_pd} \\
& + Idd3P * f_{act_pd} \\
& + Idd2F * f_{pre_standby} \\
& + Idd3N * f_{act_standby} \\
& + (Idd0 - Idd3N) * f_{activation} \\
& + (Idd4W - Idd3N) * f_{write} \\
& + (Idd4R - Idd3N) * f_{read} \\
& + (Idd5A - Idd2P) * f_{autorefresh} \\
& + Idd6 * f_{sf}) \\
& * Vdd * (Vdd_op/Vdd)^2 \\
& + (Vtt - Vtt_adj) * IOL * (nDQ + nDQS) * f_{io}
\end{aligned} \tag{3.19}$$

3.8.2 Support-Chip Power Models

The DDR DIMMs use additional support chips besides the SDRAM chips: registers for buffered access, the PLL chip for clock-generation, and an SPMD EEPROM for the serial-presence detect. This section describes the models used to calculate the power consumed by the registers and the PLL. The SPMD EEPROM dissipates very little power during normal operation and is not included in the MEMPOWER calculations.

Register Power Model

Each registered, ECC, 72-bit-wide, DDR SDRAM DIMM has register components on it that are used for buffering. Depending on the design of the DIMM, there may be two or three register parts on the DIMM. Texas Instruments [8]

provides a methodology for calculating the power dissipated by a DDR-DIMM register part. Equation 3.20 gives the total current drawn by a single register part. Power, $P_{register}$ is then the product of the current value, I_{CC} , and the voltage, V_{dd} .

$$I_{CC} = I_{CCStatic} + I_{CCClock} \times freq(clock) + I_{CCData} \times freq(clock) \times \# \text{ of data inputs} \quad (3.20)$$

The data sheets for the register part give the current values for $I_{CCStatic}$, $I_{CCClock}$ and I_{CCData} , which are used without modification. The value for $freq(clock)$ is the base, not the double-rate, DDR clock. So for DDR333 SDRAM memory, the clock value is 167 MHz, for example. The number of register data inputs used is a maximum of 13 for register parts used with stacked DIMMs and 14 for register parts used with unstacked DIMMs. Stacked DIMMs typically use 3 register parts per DIMM, and unstacked DIMMs use 2 although this is a parameter that one specifies in the DIMM specification file. Although unrealistic, the power calculations presented here assume that all of the data inputs switch on every cycle to give an upper bound on the power dissipated.

When the DIMM goes to power-down, there is no effect on the register power. When it goes to self-refresh, MEMPOWER takes the register power as 0.

PLL Power

Each DDR DIMM has a PLL part that is used to generate the clocking that the SDRAM chips require. For the PLL part, there is less in the way of explicitly defined power calculation methodology. The PLL data sheets give two current values, $IDDP_{PLL}$ and $AIDDP_{PLL}$. Based on this information and the supply voltage, V_{dd} , the PLL power is taken to be $P_{PLL} = IDDP_{PLL} \times V_{dd} + AIDDP_{PLL} \times V_{dd}$.

The PLL power is unaffected by a transition to power-down. However, when the DIMM goes to self-refresh, the PLL power is taken to be 0. Since there is still some residual support-chip power consumed in self-refresh mode, the power calculation uses a value, P_{sf} , specified in a specification file for the DIMM overhead power while self-refresh mode.

3.8.3 Summarization Calculations

Once MEMPOWER has computed the average power for the SDRAMs on a DIMM and the DIMM-level, support-chip power overhead, it calculates the total power for the DIMM and the DIMM group in the current interval using the following formulas where n_{SDRAM} is the number of SDRAMs on each DIMM and n_{DIMM} is the number of DIMMs in a DIMM group.

$$P_{overhead} = P_{sf} * f_{sf} + (n_{registers} * P_{register} + P_{PLL}) * (1 - f_{sf}) \quad (3.21)$$

$$P_{DIMM} = (P_{SDRAM} \times n_{SDRAM}) + P_{overhead} \quad (3.22)$$

$$P_{DG} = P_{DIMM} * n_{DIMM} \quad (3.23)$$

Based on the length of the interval in ordinary time units, the energy calculation is a straightforward multiplication of the power calculated for the interval times its length in ordinary time. The injected delay passes through unchanged from the utilization step.

Description	Producer	Primary Consumer(s)
Original trace	Mambo	msplitbypid, mreformat
Reformatted trace	mreformat	mspatial, minterarrivalttime
Trace with interarrival times	minterarrivalttime	mdiagram, mcountinterarrivaltimes
Diagram	mdiagram	mutilization
Interval-data	mutilization, mpower	mpower, mprintintervaldata
Interarrival-time counts	mcountinterarrivaltimes	mcdf

Table 3.4: MEMPOWER's files

3.8.4 Implementation

The `mpower` program implements the power, energy and delay calculations described above. It is a filter that reads an interval-data file produced by `mutilization` and writes an updated interval-data file containing power, energy and delay values in each interval record. The next section describes this file in more detail.

3.9 Outputs

MEMPOWER produces a variety of output information, with each stage capable of producing its own output file or set of them. Table 3.4 lists the files that MEMPOWER uses. However, for reasons of disk space and computational efficiency, most of these files are never actually written, and MEMPOWER runs many of the steps using a shell script that pipelines from one step to another.

There are two sets of files that are usually written to disk, one file of each type for each DIMM group. The first type of file summarizes information about the interarrival times of memory requests to the DIMM group, and the data are in terms of the probability mass function and cumulative distribution function of the interarrival-time random variable for the DIMM group. One uses these results to determine where to set power-management thresholds, for example, and to determine how uniformly the trace references memory across the DIMM groups.

The second, and more important, type of output file, the *interval-data* file introduced previously, is the one containing the utilization, power, energy and injected-delay data for the time intervals into which MEMPOWER divides the trace. MEMPOWER calibrates the power in milliwatts and the energy in millijoules. In addition, this file contains information about the number of cycles of delay injected into the trace by any power-management policies applied during the interval.

Each DIMM group results in a separate interval-data file that gives the utilization, injected delay, average power and total energy for the DIMM group in each interval of the trace. The post-processing programs and scripts combine the information from the interval-data files for all of the DIMM groups, for example, to give the total energy consumed by the memory during the trace.

3.10 Post-processing

Finally, there is a set of post-processing scripts and programs that MEMPOWER uses to produce human-readable output including tables and graphs.

The `mcd` program processes the memory-reference interarrival-time counts produced by `mcountinterarrival-times` and computes the probability density and cumulative distribution functions of the interarrival-time-counts random variable.

The `mprintintervaldata` program reads the binary form of the interval-data file written by `mpower` and produces an ASCII rendering of it. The remaining post-processing programs use the ASCII file, selecting portions of it, totalling the results or graphing the output. In particular, the whole-memory results are the result of post-processing programs that total across the intervals in an interval-data file and then add those results across all of the DIMM groups.

Chapter 4

Validation and Performance

Given the size of the traces and the number of processing steps, MEMPOWER's accuracy and execution performance are critically important. If MEMPOWER's calculations are too inaccurate, its results mislead its users. If it takes excessively long to run for reasonably sized traces, it has little or no value over a full simulation of the memory.

4.1 Validation

MEMPOWER's implementation raises important issues of accuracy and validation. Although relatively simple, MEMPOWER processes very large files, limiting the value of manual verification on small input streams. Moreover, the only true validation of MEMPOWER is to take power measurements on system memory while collecting a trace that MEMPOWER then processes. By comparing the results, one can determine how accurate MEMPOWER is. Since this is not generally feasible, there are two precautions taken in the implementation of MEMPOWER to attempt to ensure a reasonable level of accuracy. First, there are number of checks inserted in the code that verify the rationality of intermediate results. For example, `mpower` adds the utilization fractions in each interval to ensure that they sum to 1 before calculating the power and energy. Second, since MEMPOWER is a set of discrete processing steps, each producing an intermediate set of data structures, there are a number of separate programs that verify that there are no inconsistencies and serve as debugging aids. The most important of these include the following.

- `mprintreformatted`: renders a reformatted or clocked trace in ASCII for visual inspection.
- `mprintinterarrival`: prints the output file from `minterarrivaltime` for visual inspection.
- `mcheckdiagram`: verifies a diagram file. This program checks a diagram file to ensure that the intervals cover the temporal length of the trace, that there are no overlaps and that there are no illegal state transitions.
- `mprintdiagram`: prints a diagram file for visual inspection.
- `mcheckutilization`: checks an interval-data file produced by `mutilization` to ensure that the times assigned to the various states in each interval sum to the length of the interval.

Description	Source File	Lines of Code
Calculates the cumulative distribution of memory reference interarrival times	mcdf.c	271
Validates the diagram files produced by mdiagram	mcheckdiagram.c	235
Collects about the input trace	mcounter.c	23
Counts the interarrival-time values	mcountinterarrivaltimes.c	250
Creates an empty diagram file for totally idle DIMM groups	mcreatediagram.c	127
Produces a diagram file for a DIMM group	mdiagram.c	707
Removes all of the memory references associated with a set of PIDs from a trace	mfilteroutpid.c	200
Adds interarrival-time information to a reformatted trace	minterarrivaltime.c	200
Performs the power, energy and delay calculations	mpower.c	519
Header file for common structures and includes	mpower.h	198
Prints a trace file	mprint.c	82
Prints the maximum clock value found in a trace	mprintclock.c	68
Renders a diagram file into ASCII	mprintdiagram.c	123
Renders a reformatted trace with interarrival times added in ASCII	mprintinterarrival.c	72
Prints an interval-data file for post-processing	mprintintervaldata.c	243
Prints a reformatted trace	mprintreformatted.c	112
Reformats a Mambo-generated trace	mreformat.c	216
Determines maximum interarrival time for memory references	mscaninterarrivaltimes.c	109
Breaks a trace into smaller sub-traces	msegment.c	194
Splits a trace into sub-traces, one per DIMM group	mspatial.c	549
Parses specification files and generates binary versions of them	mspecs.c	539
Splits a trace by PID, generating one sub-trace for each PID found	msplitbypid.c	187
Optionally splits a diagram file into multiple intervals and calculates the memory utilizations in each interval	mutilization.c	549
An initial trace printing program	realmemory_bin.c	71
Header file for common structures and includes	realmemory_bin.h	112
Another initial trace printing program	realmemory_bin_parse.c	86
total		6266

Table 4.1: MEMPOWER lines-of-code sizes for the programs written in C

4.2 Implementation Considerations

All of the MEMPOWER programs except for some post-processing scripts are in C with, in most cases, bash shell wrappers to ensure consistency of execution. The following Table 4.1 lists the individual programs and their sizes in lines of code. It includes only those programs currently used by MEMPOWER in processing Mambo-generated traces and does not include the bash wrappers.

Most of the programs are filters that use standard C library functions for I/O. However, `msplitbypid` reads a single input and writes multiple output files, one per PID discovered in the trace. The `mspatial` program not only writes multiple output files but also optionally compresses them using functions from `libbzip`.

Most of the post-processing code except `mprintintervaldata` and `mcdf` consists of either Perl or R scripts. The Perl scripts are used for table preparation and reformatting while the R scripts graph the results for presentation. Currently, the post-processing done with MEMPOWER does not make use of the statistical capabilities of R [9].

4.3 MEMPOWER's Performance

The goal of MEMPOWER is to run relatively quickly when processing a trace of significant size. Although there has never been a formal evaluation of MEMPOWER's performance, anecdotal evidence suggests that it meets this goal, and most of the performance problems encountered in running MEMPOWER are associated with the amount of disk space consumed by the input traces and the data that MEMPOWER generates.

Table 4.2 summarizes the characteristics of the Mambo-generated trace discussed in Chapter 5. When processing

Trace size in bytes as compressed by bzip2	528790191
Trace size in bytes	6043055280
Number of trace records	302152764
Length of the trace in 2 GHz processor clocks	1205007557625
Length of the trace in seconds	561.125370
Number of memory references	301344968
Number of memory reads	277681544
Number of memory writes	23663424

Table 4.2: Trace statistics for the test trace

this trace to produce the results in Chapter 5, MEMPOWER ran on an IBM IntelliStation Z Pro machine with two (2) processors, executing as four (4) by means of hyperthreading, 2 GB of main memory, and two (2) local, 36 GB, 15000 RPM, SCSI disks. The machine uses the Fedora Core 1 distribution of Linux, and all data is accessed from the local disks. To reduce the consumption of disk space, MEMPOWER stored all of its data files in compressed form and used a pipeline to run the `mdiagram`, `mcheckdiagram`, `mutilization` and `mpower` sequence of programs. Total processing time, prior to post-processing, was less than 48 hours with some idle stretches overnight.

In considering the reasons why MEMPOWER takes as much time as it does, it is apparent that one of the most vexing problems with MEMPOWER is that every power-management policy considered requires that one run a sequence of programs – `mdiagram`, `mutilization` and `mpower`, usually in a pipeline – for every DIMM group and potentially for every PID and every DIMM group. This means that the DIMM group interarrival-time file is read repeatedly. It also leads to a combinatorial explosion in the number of runs needed when one is exploring the space of power-management options.

Chapter 5

A Sample MEMPOWER Analysis

This chapter presents a MEMPOWER analysis of two Mambo-generated [3] traces of the TPC-W benchmark. The power results shown here are illustrative of what MEMPOWER can produce and include whole memory, per-DIMM-group and per-DIMM-group and per-process power and energy values under a number of different power-management policies. The goal of this particular study is to determine the best power-management policy or combination of policies for the memory controllers in a system with power-aware virtual memory(PAVM) [6] running a transaction-processing workload [10]. The results presented here form a subset of a larger study reported in the work of Huang, et al, [2]. That paper also provides more details on the power management policies mentioned here and their motivation.

5.1 Benchmark

The benchmark used in generating the trace is based on the TPC-W e-commerce specification. The traces are from runs using

- Linux 2.4.19
- Apache
- PHP
- MySql
- an implementation of the TPC-W logic originally done at Rice University [10].

The Linux 2.4.19 is a PowerPC version with an implementation of PAVM ported from the x86 environment. For comparison purposes, the study also references a separately generated trace that uses a standard Linux 2.4.19 kernel without the PAVM enhancement. The trace is for 20 TPC-W clients and is nominally 600 seconds in length. The assumed processor speed is 2 GHz with 1 GB of memory, and the machine simulated is a uniprocessor system. MEMPOWER divides the memory into 32 DIMM groups with an interleaving factor of one (1), so that it effectively ignores interleaving in this case. MEMPOWER treats each DIMM group as having a single DIMM.

5.2 Memory Layout and Parameters

As mentioned previously, MEMPOWER gets the information about currents and voltages from the specification files that it processes. However, that data, in turn, comes from the data sheets provided by the parts vendors. In particular, MEMPOWER requires voltage and current data that the memory vendors provide in their datasheet documents. The results presented here use datasheet values for the Micron 64 megabit DDR SDRAM parts using the DDR333 speed. These are packaged on ECC registered DIMMs, with five (5) DRAMs, two (2) registers and one (1) PLL on each DIMM. For a variety of business reasons, the currents quoted in the vendor’s data sheets are often significantly higher than those observed in practice. To model this feature, MEMPOWER has the ability to scale the data sheet currents: the scaling factor is a parameter that the user must set based on experience. In processing these results, it is set to 0.8.

5.3 Trace and Power-Management Characteristics

The trace used here is, unfortunately, not as memory-intensive as one might like, so that relatively simple power-management policies do quite well in terms of reducing the overall memory power. Table 4.2 summarizes some of its characteristics. This study considers four general policies for power management beyond PAVM – none, power-down after an idleness threshold, self-refresh after an idleness threshold and a combination policy under which the memory goes to power-down as soon as it goes idle and then goes to self-refresh after a threshold. Much of the effort in the evaluation is directed toward determining how to set the threshold values.

5.4 Interarrival Times

One important question for memory power-management policies is whether the system uses the memory uniformly across the entire range of physical addresses. For the TPC-W trace, it does not. The results from the MEMPOWER interarrival-time processing shown in Figure 5.1 demonstrate this. Figure 5.1 graphs the cumulative distribution functions for DIMM groups (labelled here as “memory modules”) 0, 2 and 24.

5.5 Whole Memory Results

One of the most important comparisons is the amount of energy expended by the entire memory in the course of the whole run. Figure 5.2 compares the total energy over the course of the run for entire memory using different policies. The cooperative policy shown here uses power-down, followed by self-refresh with different thresholds for entry to self-refresh for different processes in the trace. These results are illustrative of the ability of MEMPOWER to analyze traces rather than a final evaluation of any architectural ideas about how to manage memory power.

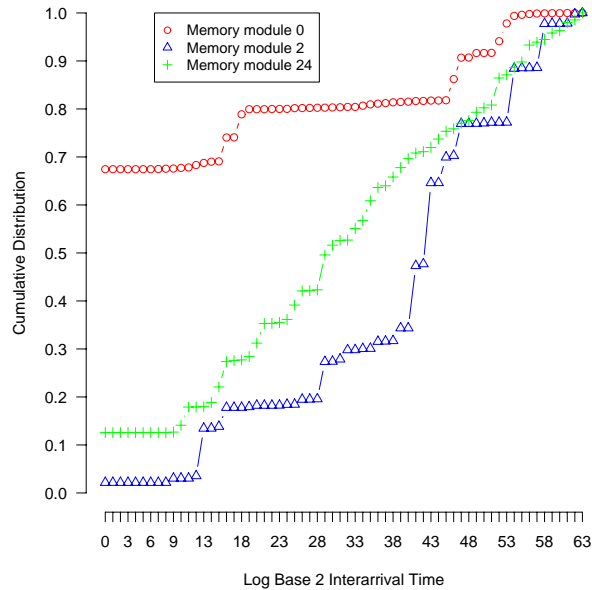


Figure 5.1: The cumulative distributions of the memory-reference interarrival times for DIMM groups 0, 2 and 24

5.6 Results for Individual DIMM Groups

Figure 5.3 shows the normalized energy consumed under different power-management policies while Figure 5.4 presents the injected delay normalized to the temporal length of the original trace.

5.7 Per-Process Power Consumption

To illustrate the results that MEMPOWER can produce for individual processes, Figure 5.5 shows the difference in normalized energy consumption for DIMM group 0 by two processes using a combined power-down and self-refresh power management policy with different self-refresh thresholds.

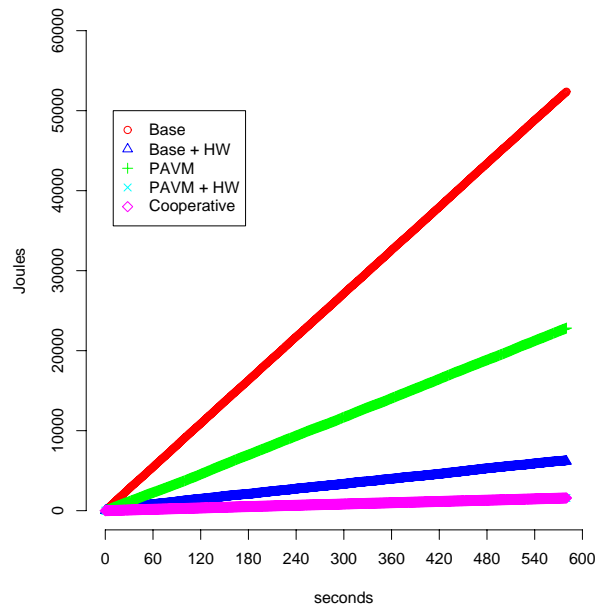


Figure 5.2: Comparison of memory energies for the entire memory over the course of the TPC-W trace using different power-management policies. The number of visible lines is smaller than the number of cases due to the fact that the energies for PAVM+HW and Cooperative overlay each other.

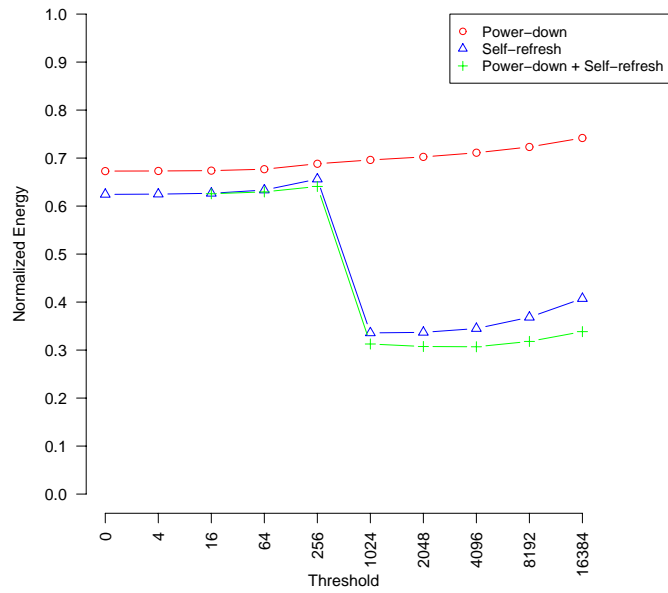


Figure 5.3: DIMM group 0 normalized energy under different power-management policies

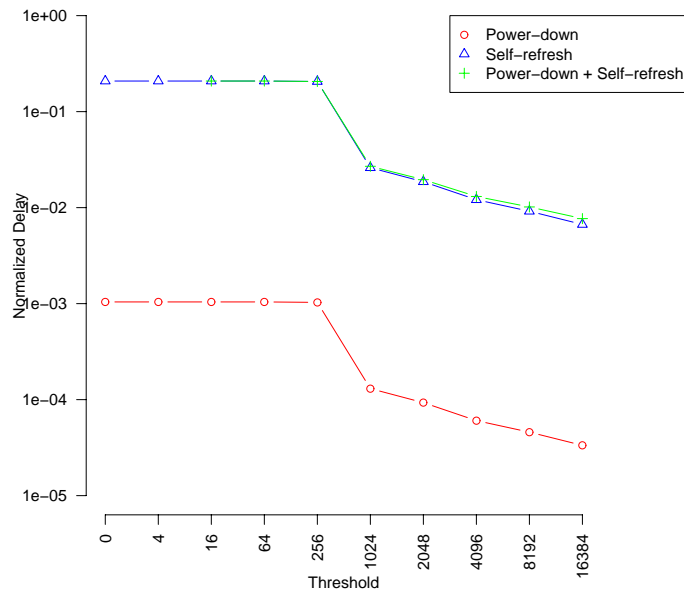


Figure 5.4: DIMM group 0 normalized delay under different power-management policies

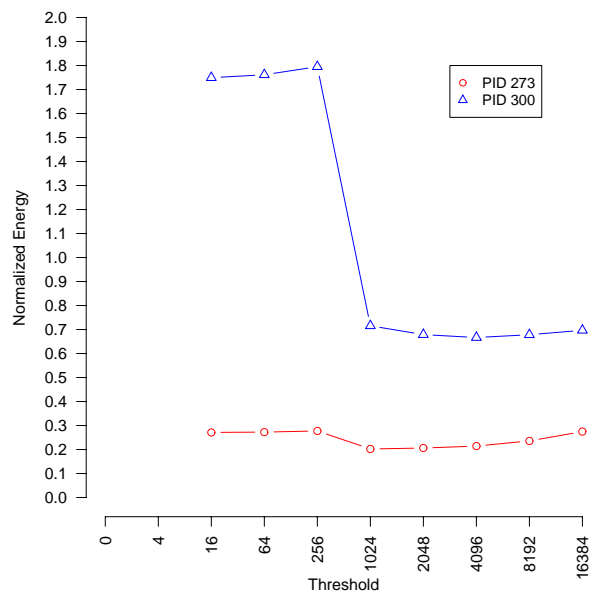


Figure 5.5: DIMM group 0 normalized energy for two different processes with different self-refresh thresholds and a combination power-management policy

Chapter 6

Future Work

The schedule for MEMSIM determines how much future investment should be directed to MEMPOWER. If MEMSIM is going to be ready in the near future, there is little reason to continue developing MEMPOWER. However, if there will be a substantial delay before MEMSIM is generally usable, there is good reason to continue investing in MEMPOWER.

Currently, MEMPOWER has two major implementation deficiencies. First, it is difficult for anyone but the author to use since it requires using parameters and scripts to run a relatively large number of programs that must all execute in the proper sequence. This problem is best solved by adding a Java-based graphical user-interface that allows the user to enter the necessary parameters and then runs the programs in the correct order. This interface can also integrate the data-reduction and visualization phases that one must currently do largely by hand. Second, there is a performance problem due to the combinatorial explosion in the number of runs required to explore a set of power-management policies. Exploring the threshold space in parallel in `mdiagram` may help solve this problem. However, the logic to do the parallel exploration is reasonably complex.

There are a number of other extensions possible beyond correcting these two deficiencies in the implementation. The current version of MEMPOWER takes traces generated by Mambo simulations of uniprocessor systems, and there is no processor identification information in the trace. An SMP simulation can produce a trace that tags each memory reference with the processor issuing it as well as tagging all of the PID-related records with a processor identifier as well. MEMPOWER can use that information to determine the memory power associated with either a particular processor or a particular process running in an SMP environment where processes may move from one processor to another.

The next generation of memory technology is beginning to appear in system designs. DDR-2 SDRAMs have a slightly different set of operating states and associated currents than DDR. Properly calculating memory power for DDR-2 requires a slight modification of the `mpower` program. RAMBUS memory also has a different set of operating states and currents as well as a more complex set of power management modes: supporting RAMBUS requires noticeably more effort than just adding DDR-2.

MEMPOWER needs more extensive use on memory traces of a variety of workloads. One can experiment with

different service times and delays in the memory system, different current values for different parts and different traces of other workloads.

Finally, there is the challenge of validation. In an optimal environment, one would do the validation work described in Chapter 4 to ensure that MEMPOWER is yielding accurate results.

Acknowledgments

Hai Huang and Eric VanHensbergen collected the traces whose evaluation appears here. Hai Huang developed PAVM at the University of Michigan and ported it to PowerPC Linux while an intern at the IBM Austin Research Laboratory during the summer of 2003. Pat Bohrer, Jim Peterson, Eric VanHensbergen, Hazim Shafi and many others contributed to the development of the Mambo full-system simulator. Karthick Rajamani contributed to related work on memory-power measurement and analysis that helped to develop the theory used by MEMPOWER. He also wrote the TPC-W implementation and is developing MEMSIM to replace MEMPOWER. Sudhanva Gurumurthy was the summer student who started the MEMSIM implementation. Tom Keller and Mootaz Elnozahy managed the project, and Mike Rosenfield supported it as the director of the IBM Austin Research Laboratory. The project was funded by IBM's Research Division, Systems Group and Energy-Efficiency Institute.

Bibliography

- [1] Micron Technology, Inc., “Calculating Memory System Power for DDR,” Tech. Rep. TN-46-03, May 2001.
- [2] H. Huang, E. Van Hensbergen, F. Rawson, T. Keller, and K. G. Shinn, “Cooperative software-hardware power management for DRAM,” in *Submitted to 31st Annual International Symposium on Computer Architecture ISCA 2004*, 2004.
- [3] H. Shafi, P. J. Bohrer, J. Phelan, C. A. Rusu, and J. L. Peterson, “Design and validation of a performance and power simulator for PowerPC systems,” *IBM Journal of Research and Development*, vol. 47, no. 5/6, 2003.
- [4] S. Gurumurthi and K. Rajamani, “MEMSIM: Main-memory system simulation of server machines,” in *Submitted to ACM SIGMETRICS – Performance 2004*, 2004.
- [5] J. M. Tendler, J. S. Dodson, J. S. Fields, H. Le, and B. Sinharoy, “POWER4 system microarchitecture,” *IBM Journal of Research and Development*, vol. 46, no. 1, pp. 5 – 26, 2002.
- [6] H. Huang, P. Pillai, and K. G. Shin, “Design and implementation of power-aware virtual memory,” in *USENIX 2003 Annual Technical Conference*, pp. 57–70, June 2003.
- [7] Rambus Corporation, “Rambus technology overview,” Feb. 1999.
- [8] Texas Instruments, Inc., “Low-Power Support Using Texas Instruments SN74SSTV16857 and SN74SSTV16859 DDR-DIMM Registers.” <http://focus.ti.com/docs/apps/catalog/resources/appnoteabstract.jhtml?abstractName=scea020>, February 2001.
- [9] “R Project for Statistical Computing,” 2004.
- [10] C. Amza, E. Cecchet, A. Chanda, A. L. Cox, S. Elnikety, R. Gil, J. Marguerite, K. Rajamani, and W. Zwaenepoel, “Specification and Implementation of Dynamic Web Site Benchmarks,” in *IEEE 5th Annual Workshop on Workload Characterization (WWC-5)*, November 2002.