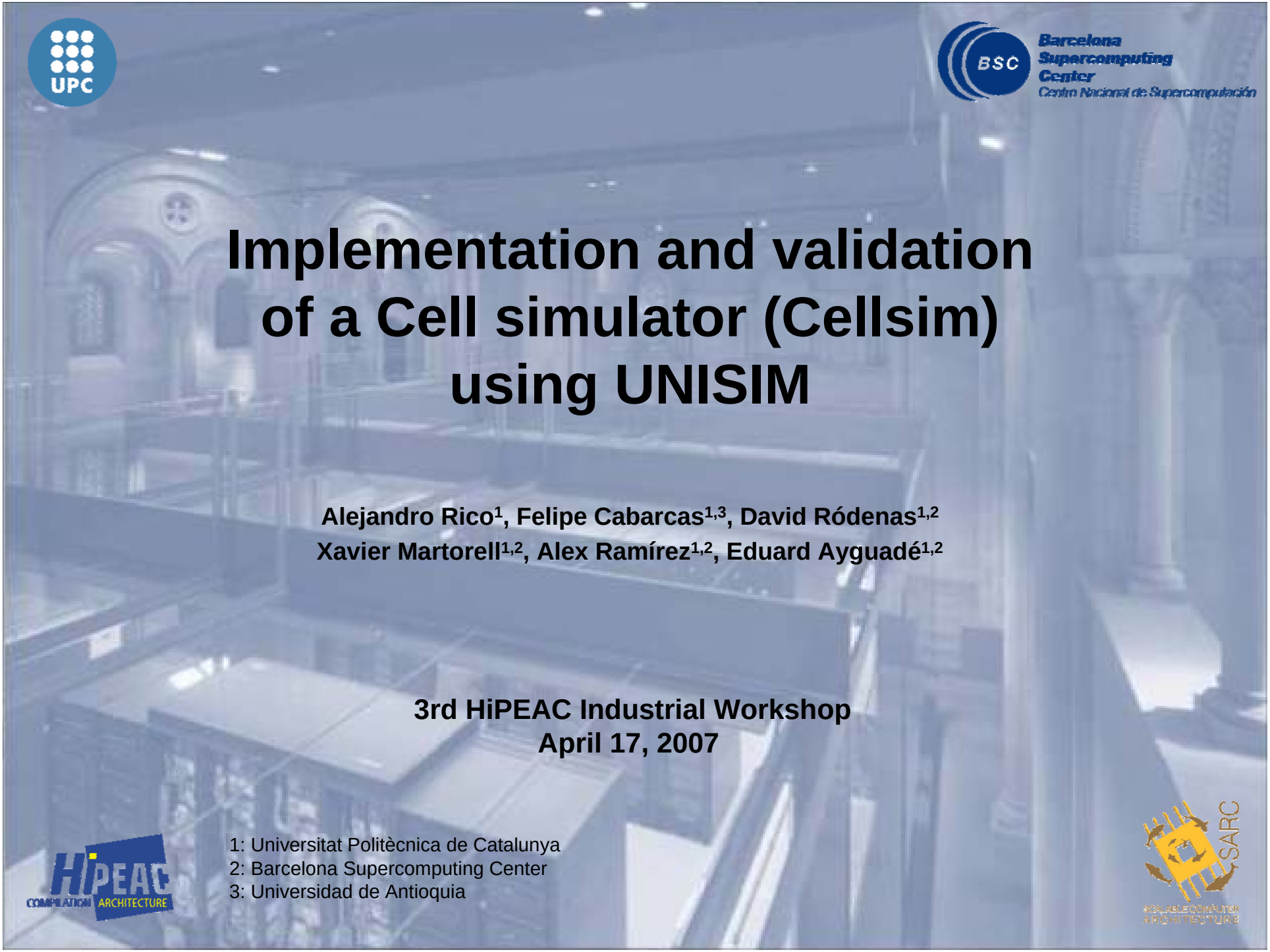


Implementation and validation of a Cell simulator (Cellsim) using UNISIM

Alejandro Rico[1], Felipe Cabarcas[1,3], David Ródenas[1,2]
Xavier Martorell[1,2], Alex Ramírez[1,2], Eduard Ayguadé[1,2]

3rd HiPEAC Industrial Workshop
April 17, 2007

1: Universitat Politècnica de Catalunya
2: Barcelona Supercomputing Center
3: Universidad de Antioquia

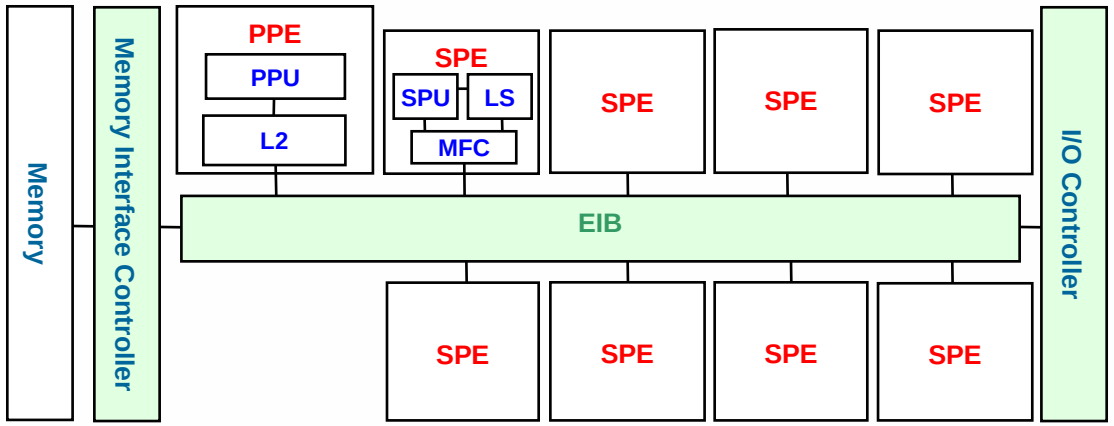
Outline

- Cellsim structure
- Modules description
- Validation
- Cellsim performance

Outline

- **Cellsim structure**
  - **Cellsim block diagram**
  - **Infrastructure**
  - **Communication of modules**
  - **Simulation process**
- Modules description
- Validation
- Cellsim performance

Cellsim and Cell block diagram

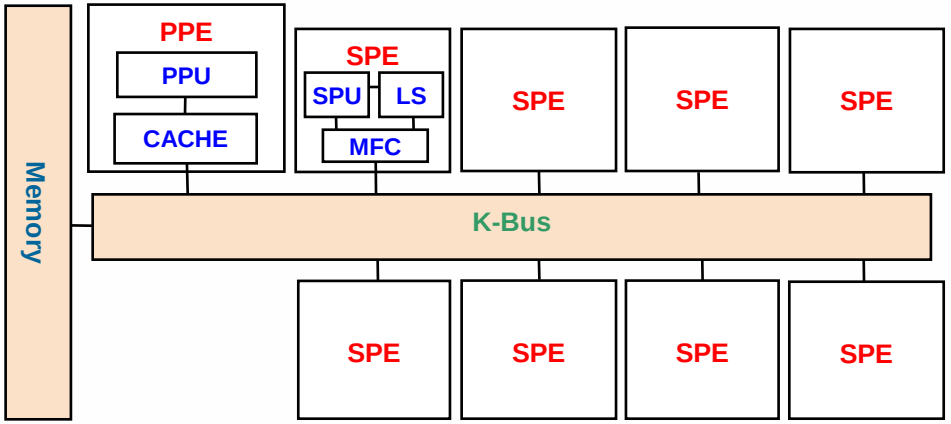


Cell HW components

- 1 PPE
- 8 SPEs
- 4-ring EIB
- 1 MIC
- 1 I/O

Cellsim modules

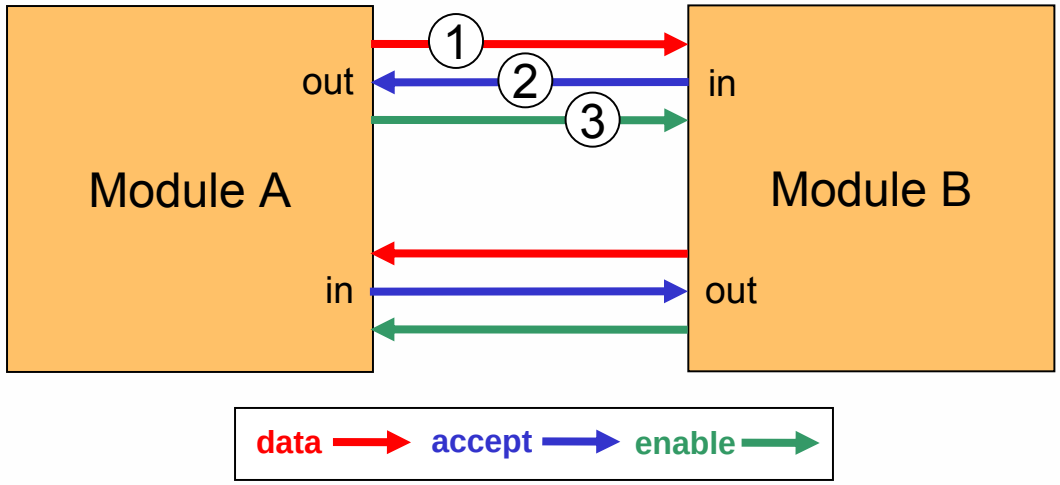
- PPE compliant.
- SPE compliant.
- The K-bus simulates the interconnection network.
- The memory is directly connected to the bus.



Infrastructure

- Cellsim is built using UNISIM framework (http://unisim.org).
  - UNISIM allows to define an architecture by creating a set of modules and their connections.
- Each Cell hardware component is mapped to a UNISIM module.
- Cellsim is being developed using Cycle-level UNISIM version.
  - All modules synchronize each cycle.

Communication of modules



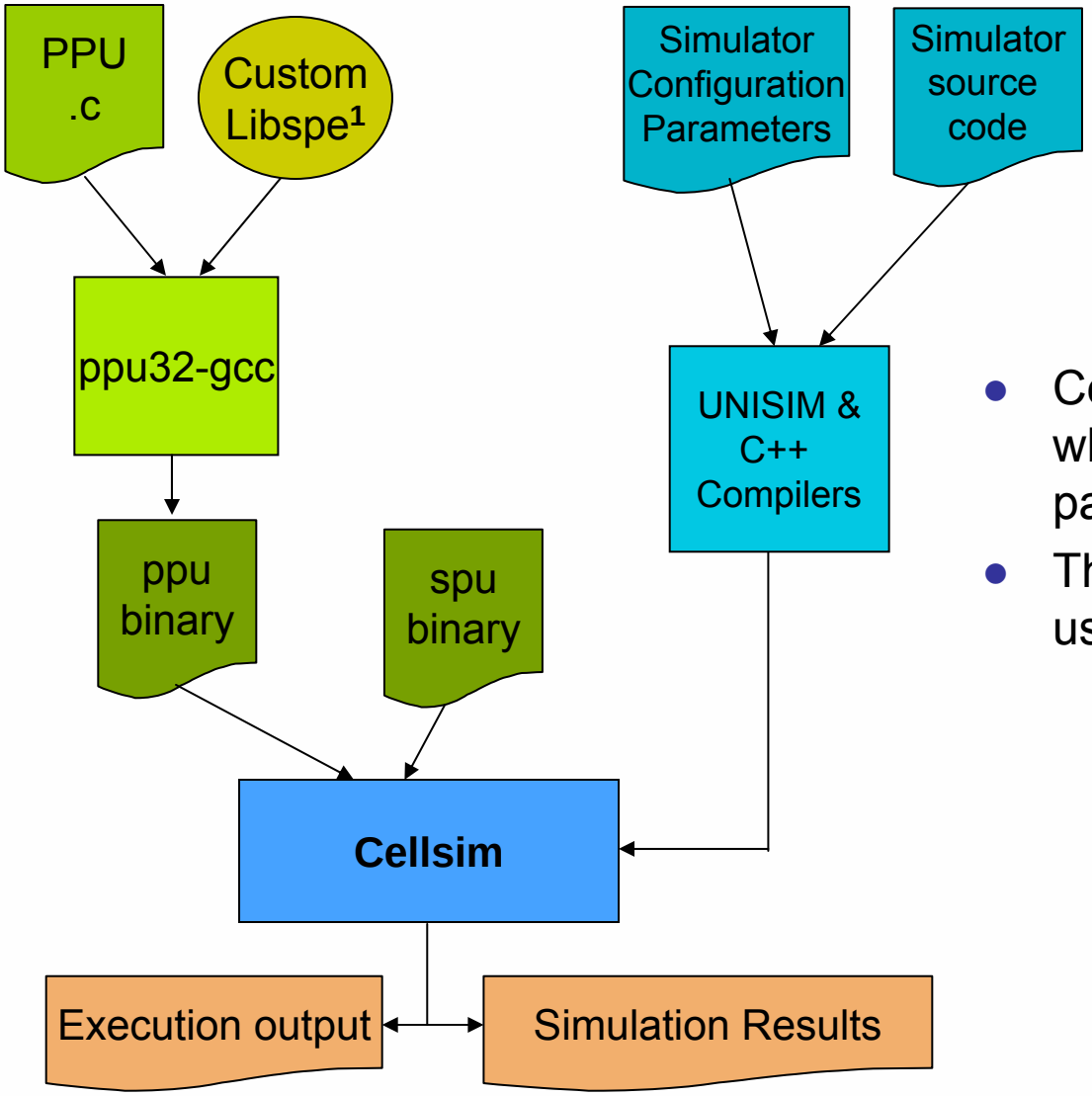
UNISIM Communication protocol

Each cycle:

1. **A** sends message.
2. **B** sets the accept signal to *true* or *false*
3. Finally, **A** sets the enable signal to *true* or *false*

- A **common interface** was developed for all module connections.
  - A message packet called MemoryAccess.
- MemoryAccess class main characteristics
  - Target and Source addresses.
  - Access type: LOAD or STORE.
  - Data
- This common interface allows the reusability and sharing of modules.

Simulation process



- Cellsim should be recompiled when the configuration parameters are changed.
- The PPU code has to be compiled using a *custom libspe*.

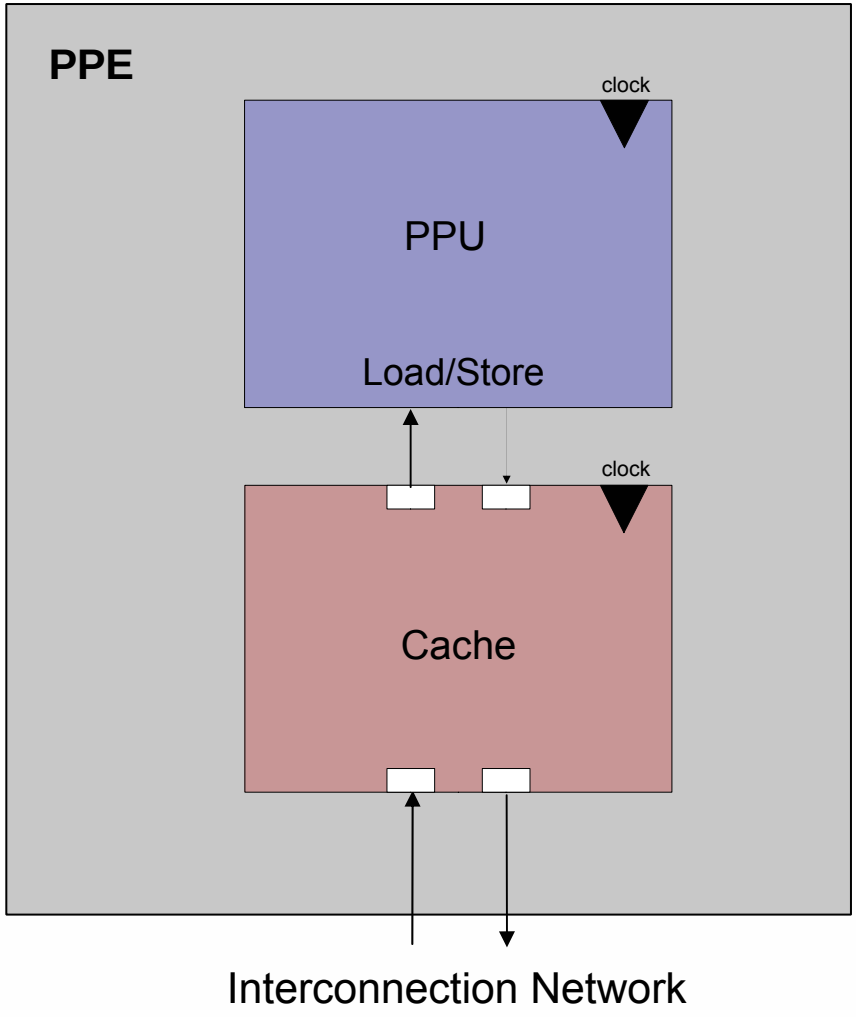
1. PPU library for SPE management

Outline

- Cellsim Structure
- **Modules description**
  - **PPE**
  - **SPE**
  - **Memory**
  - **Interconnection Network**
- Validation
- Cellsim performance

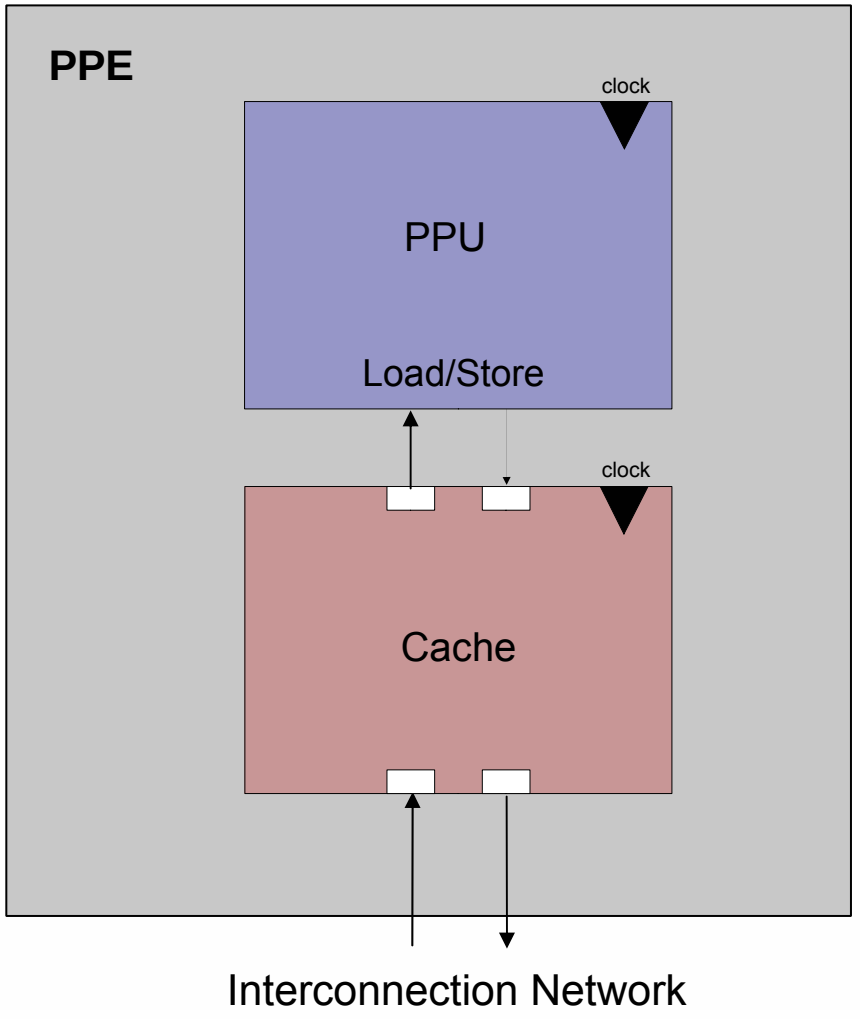
PPE structure and interface

- Structure
  - PPU
    - 32-bit PowerPC architecture compliant.
    - In order functional simulator.
  - Cache
    - Write through policy.
    - One access per cycle.
- Interface
  - PPU
    - Loads/Stores
  - Cache
    - Cache misses and accesses to other devices travel to/from the interconnection network.

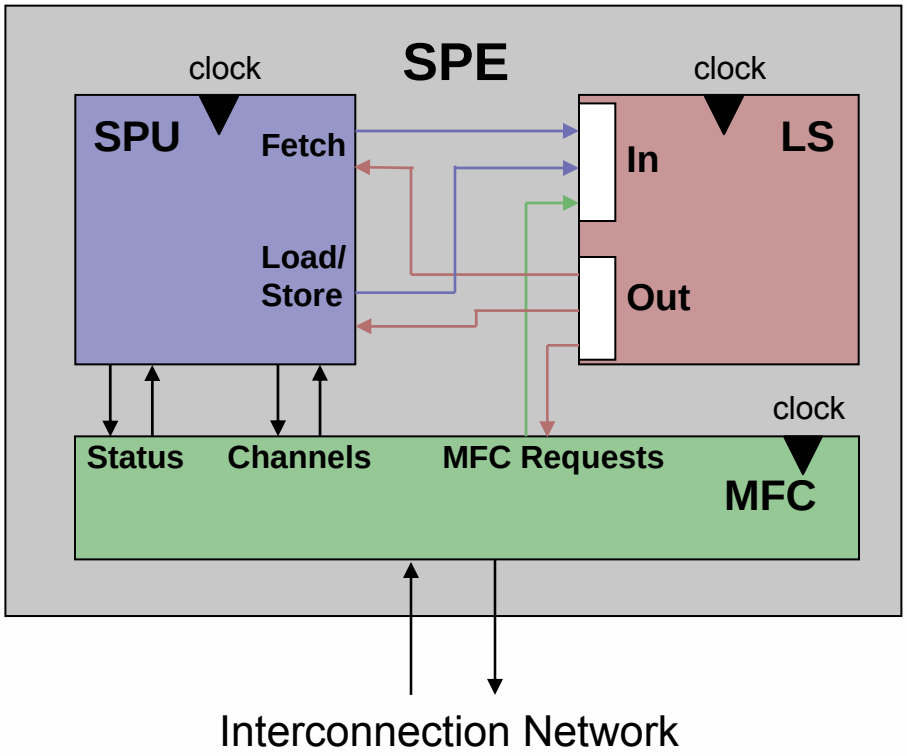


PPE configuration parameters

- PPU
  - Issue bandwidth.
- Cache
  - Number of lines.
  - Line size.
  - Number of ways.

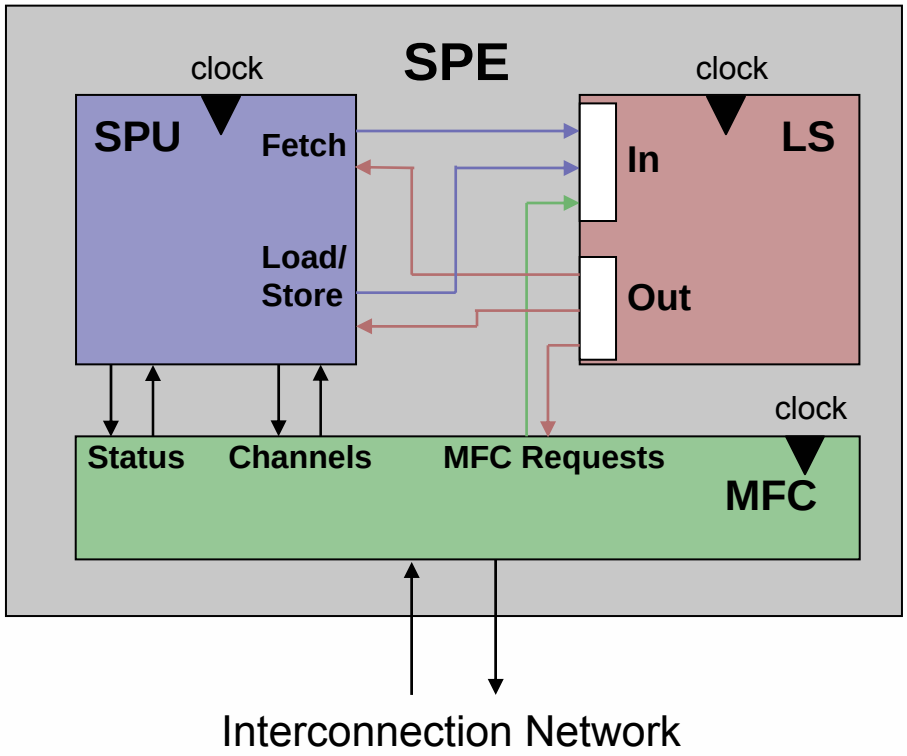


SPE structure and interface



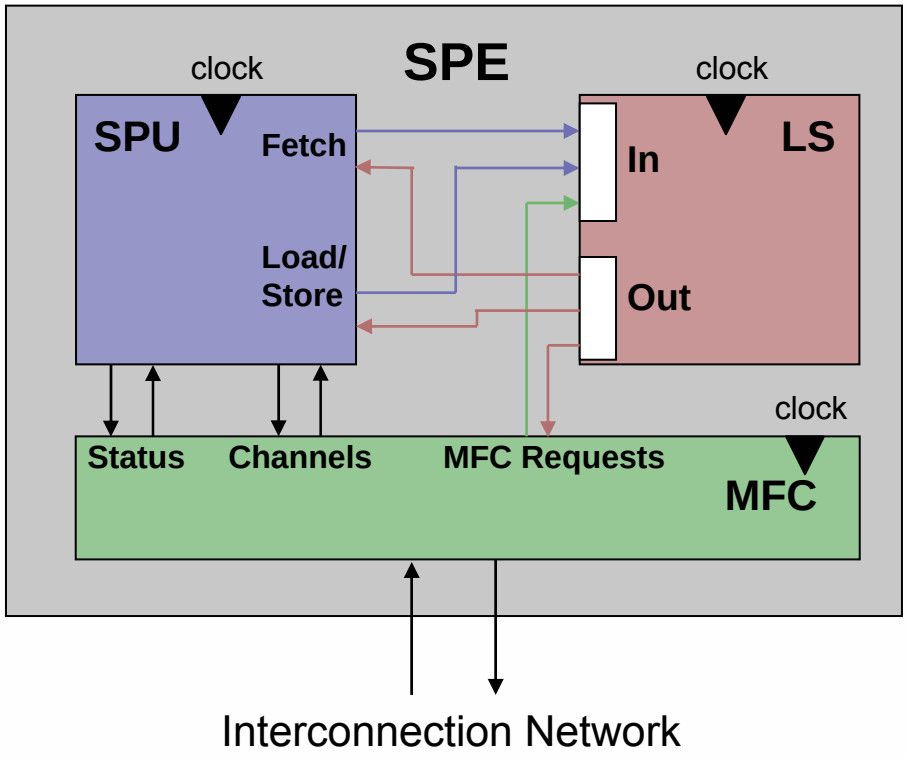
- SPU
  - **Fetch**
  - **Load/Store**
  - **Channels (read/write)**
  - **Status**: updates MFC status register.
- LS
  - **In**: receives loads and stores.
  - **Out**: sends data for received loads.
- MFC
  - **Status**: Starts/Stops SPU.
  - **Channels**: responds requests.
  - **MFC Requests**: loads and stores of DMA Commands and MMIO accesses.

SPE behavior



- SPU
  - Functional simulator.
- MFC
  - Functional simulator.
  - Executes DMA commands in order.
  - Serves MMIO accesses to MFC registers and LS.

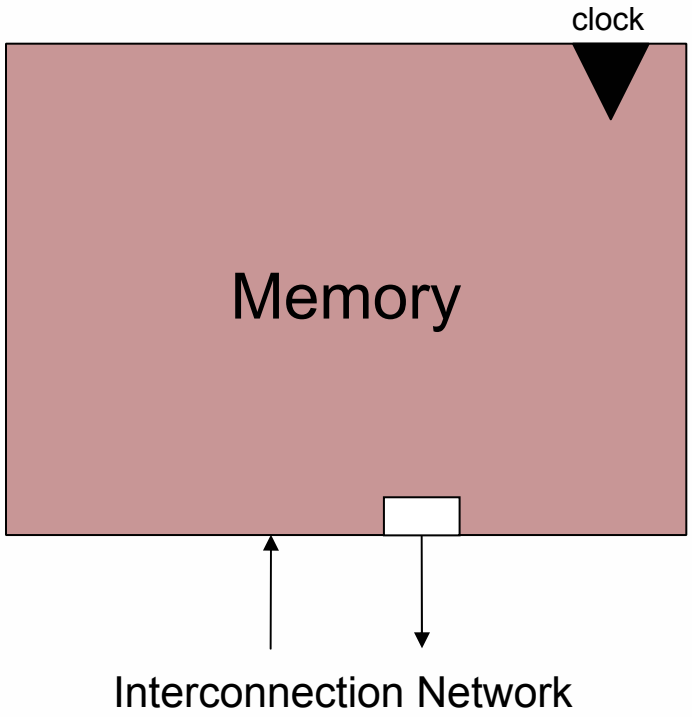
SPE configuration parameters



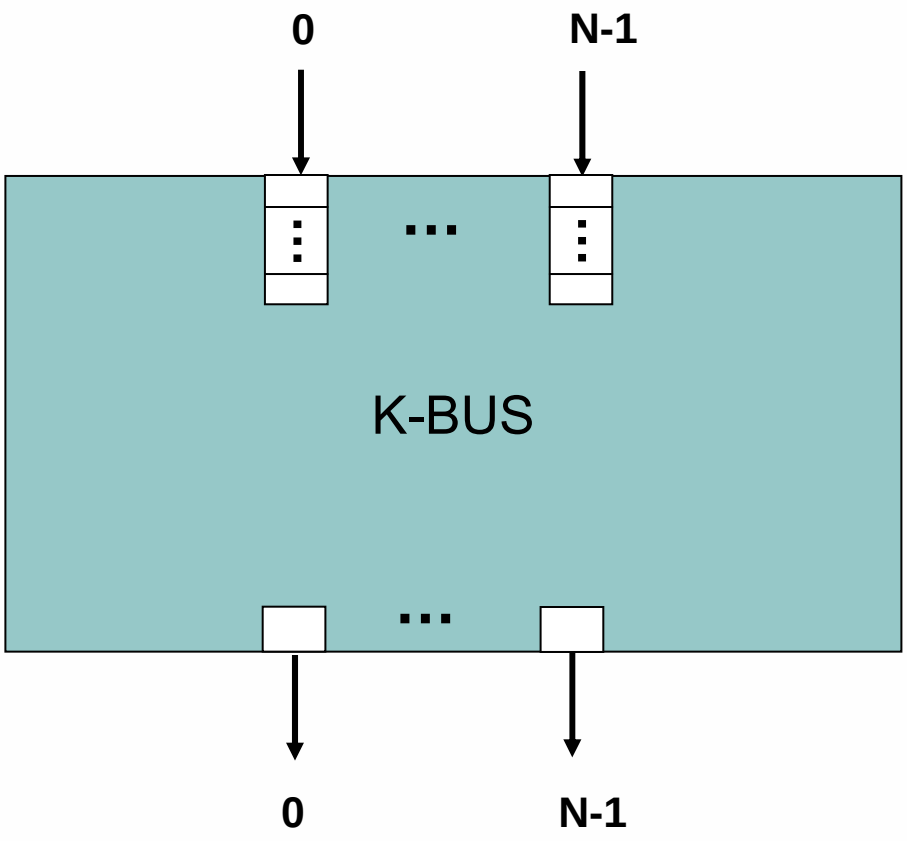
- SPU
  - Issue bandwidth.
- LS
  - Size
  - Latency
  - Number of ports.
- MFC
  - DMA command queue size.
  - DMA command processing delay.

Memory

- Interface
  - Loads/Stores coming from the Interconnection Network
- Behavior
  - Unlimited bandwidth.
  - Currently, one cycle latency.
- Configuration parameters
  - Memory Size

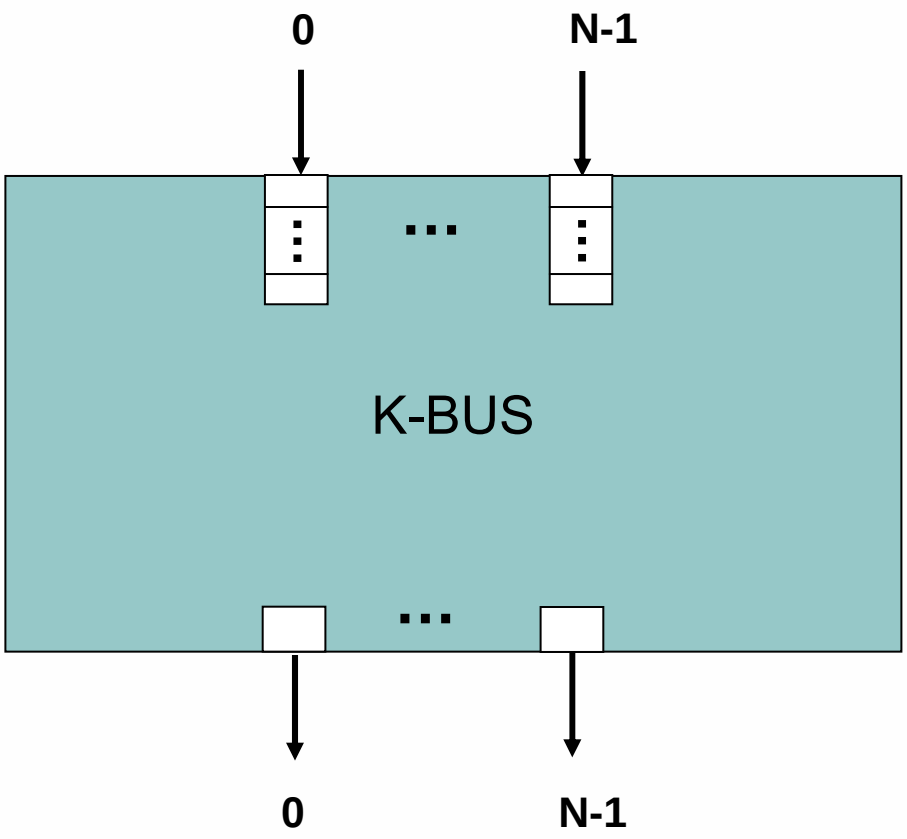


Interconnection network structure and interface



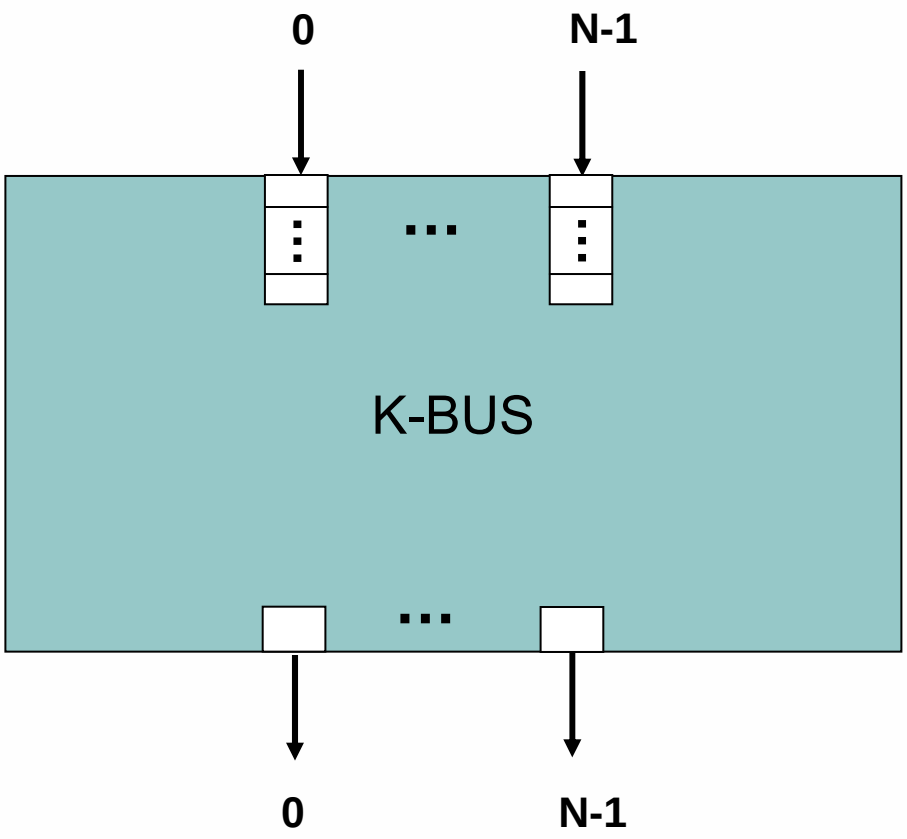
- The interconnection network is composed of K buses.
- Each port has multiple outstanding transfers.
- Port mapping:
  - Port 0: Memory.
  - Ports 1 to NP: PPEs.
  - Ports NP+1 to N-1: SPEs.

Interconnection network behavior



- Routes Accesses
- Simulates latency of transfers and the contention.
- Bus priority:
  1. Memory
  2. PPEs (round-robin among them)
  3. SPEs (round-robin among them)
- Memory coherence:
  - SPE stores to Memory are notified to the Caches for line invalidation.

Interconnection network configuration parameters



- Number of buses.
- Bandwidth
- Number of outstanding transfers per node.

Outline

- Cellsim Structure
- Modules description
- **Validation**
  - **PPU**
  - **SPU**
  - **MFC and Interconnection Network**
- Cellsim performance

PPU functional validation

- There are 183 instructions of the PPU ISA implemented.
  - Floating Point, VMX, atomic and system administration instructions are not supported.
- Methodology and tools
  - GDB[1] was used to generate a register state trace in the real machine for all programs.
  - The traces were compared with the ones generated by Cellsim.
- Programs
  - Hand made programs.
  - SDK LibC was also used.

Programs	Description	Executed Instructions	
		PPU	SPU 0
exit	Starts a program up, initializes LibC and calls to systemcall exit(0)	3978	-
getpagesize	Check for correct kernel arguments passed to LibC and uses printf to display the result of getpagesize	5532	-
vecmultO3	Scalar vector multiplication with optimization flag to O3 and posterior verification of the result	110502	-
ppe-dma	The PPU initiates a DMA Get command in a SPE. The SPE checks the received data.	14512	479

1. GNU Project Debugger (http://sourceware.org/gdb/.)

SPU functional validation

- There are 138 instructions of the SPU ISA implemented
- Methodology
  - The state of the registers and the LS was compared with the IBM's Cell Simulator one.
- Programs
  - Hand made programs: e.g. matrix multiplication, random generator.
  - Some SDK simple examples modified: e.g. spu_clean, simpleDMA.
  - The programs used to validate the MFC have also helped with the SPU ISA validation.
  - IBM's SDK tests for SPU's intrinsics (cell-sdk-1.1/src/tests/intrinsics).

Program	Description	Executed instructions		
		SPU 0	SPU 1	SPU 2
spu_clean	The SPU clears the LS (even its code) and the registers.	19684	-	-
MatMult	Two SPUs compute a 64x64 elem matrix multiplication (one using intrinsics), the third compares the results.	4285094	381742	66216
simpleDMA	The PPU generates a fibonacci sequence. The SPU gets it with a DMA command and checks its correctness.	2063	-	-
scanf	A *scanf* from the SPU code.	5365	-	-

MFC functional validation

- Methodology
  - The programs results were compared against the IBM's Simulator and the real machine (registers state, local store and main memory.)
- Programs
  - Simple hand made programs that exploit concrete functionalities (Channels, MMIO Registers, Mailboxes, Signals, DMA-transfers, DMA List transfers.)
- Atomic, synchronization and storage control commands are not supported yet.

Program	Description	Executed instructions		
		SPU 0	SPU 1	SPU 2
mailbox	The SPU writes and reads to/from the mailboxes.	95	-	-
signals	SPU 2 receives an overwritten signal from the PPU. SPU 1 receives an ORed signal from the PPU. SPU 0 receives a signal from SPU 1.	87	169	106
dma-list	A SPE executes a GETL (Get using lists) command. Another executes a PUTL.	7572	8605	-
chcount	The SPU tests the *count* of blocking channels.	190465	-	-

MFC and Interconnection Network performance validation

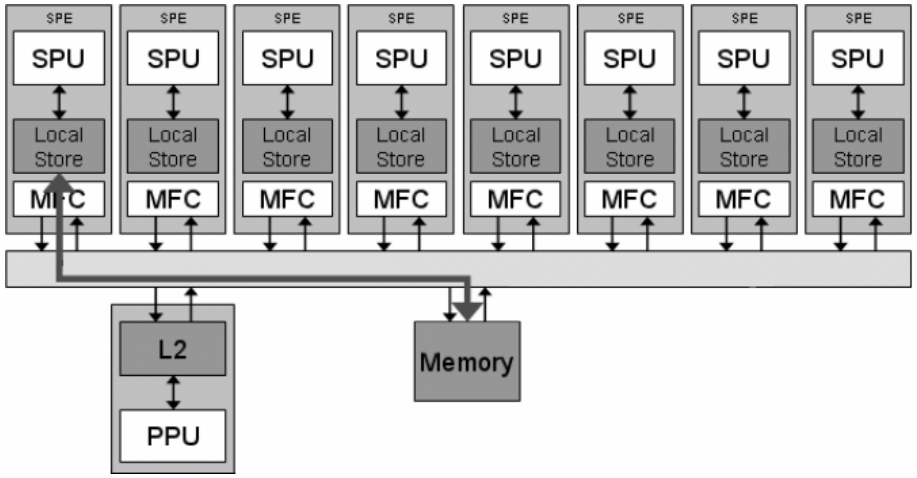
- The programs used by Daniel Jimenez's EIB bandwidth study [1], were selected for the validation.
- These programs stress the interconnection network by performing continuous DMA operations to main memory or between SPEs.
- They measure the number of decrementer *tics* it takes to perform a number of operations.
- The Cellsim results have been compared to those of the actual machine.

The frequency of the Cell used is 2.4 GHz

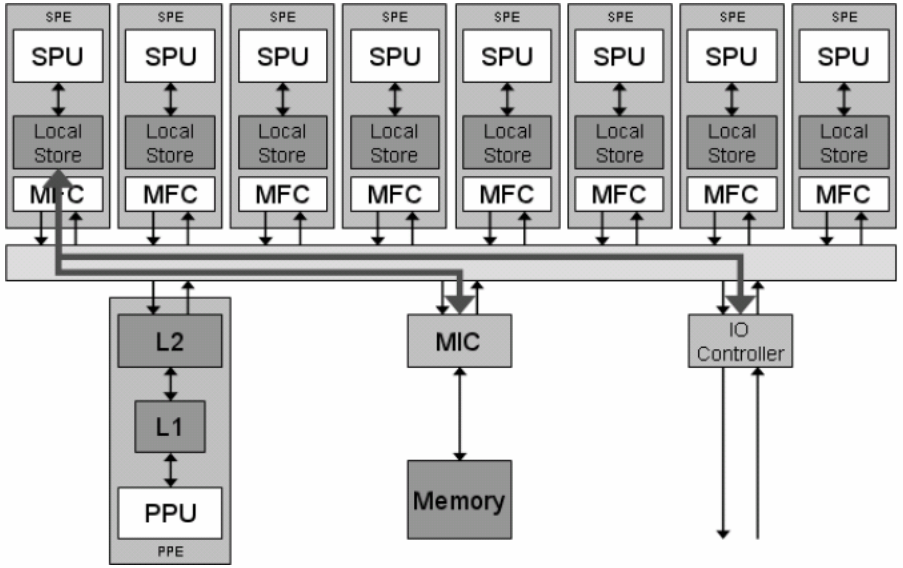
	Number of PPEs	Number of SPEs	Number of buses	BW
Cellsim Parameters	1	8	4	16 bytes / bus cycle

SPEs communication with memory validation program

- SPE-MEM: the SPEs perform DMA transfers to memory
  - Each SPE sends and receives a total of 32MBytes of data with DMA operations of the following sizes: 64, 128, 256, 512, 1024, 2048, 4096, 8192 and 16384 bytes.



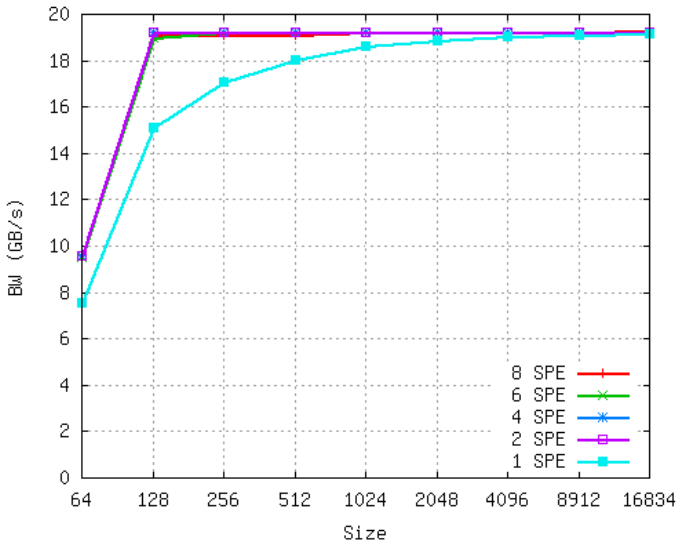
Cellsim Experiment



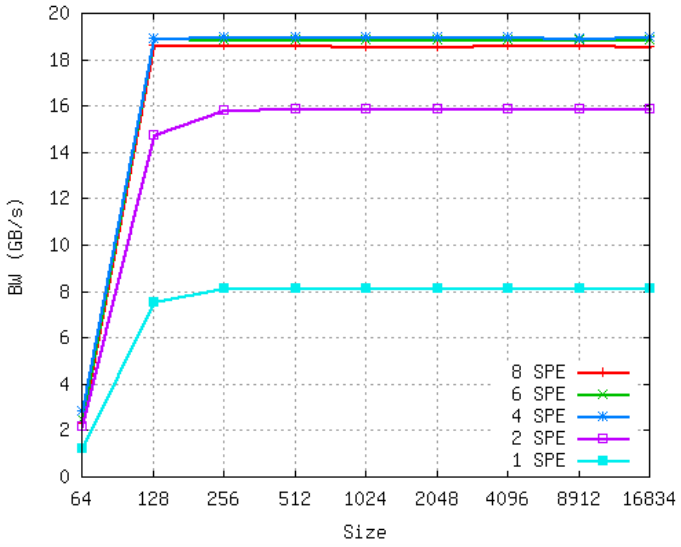
Cell Experiment

Interconnection Network BW vs. Size of transfers (SPE-MEM)

Cellsim



Cell



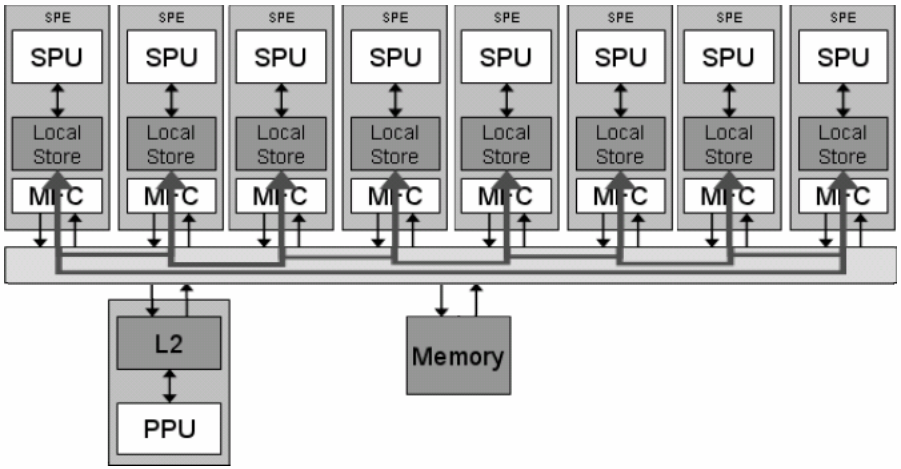
- The Cellsim bandwidth is limited by the connection to main memory (currently, the simulator memory has unlimited bandwidth.)
- Therefore, the Cellsim peak bandwidth is:

16 bytes/bus cycle * 1.2 GHz = 19.2 GB/s

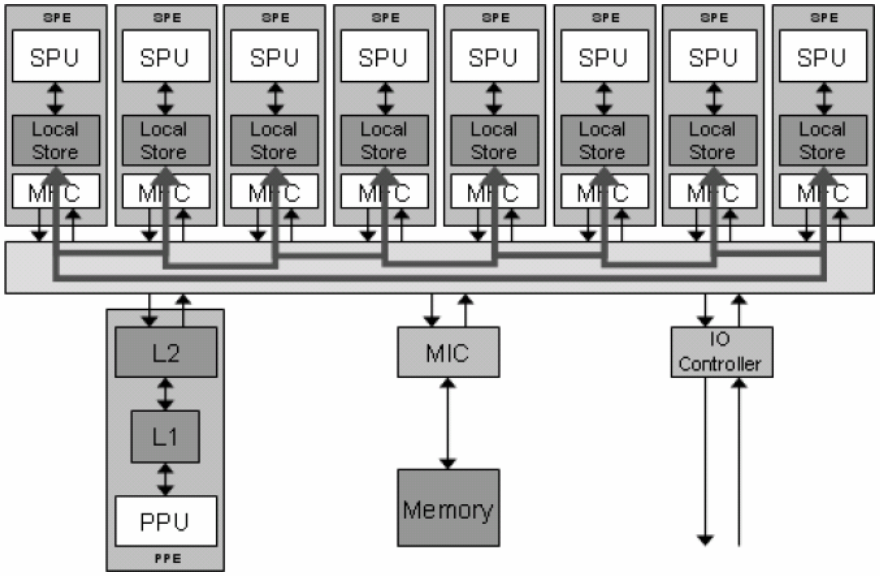
- The Cellsim achieves better performance than the Cell, for one and two SPEs, because it does not simulate the Memory Interface Controller.

SPEs communication in cycle layout validation program

- SPE-SPE cycle: each SPE *i* performs DMA transfers to SPE *i+1*
  - Each SPE sends and receives a total of 32MBytes of data with DMA operations of the following sizes: 64, 128, 256, 512, 1024, 2048, 4096, 8192 and 16384 bytes.



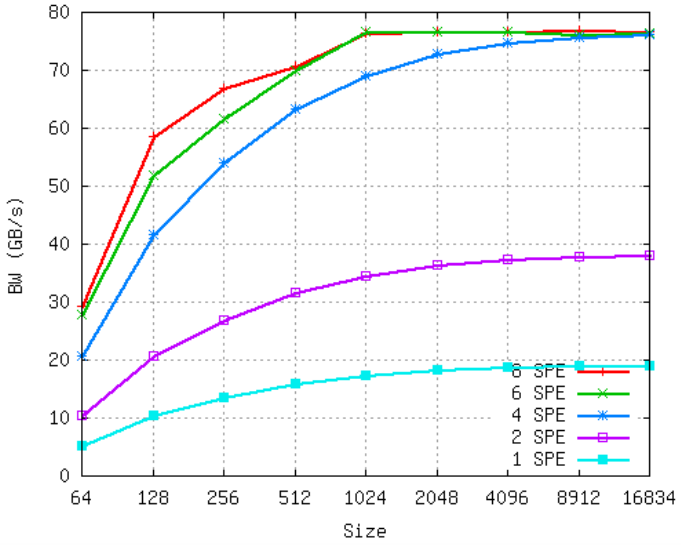
Cellsim Experiment



Cell Experiment

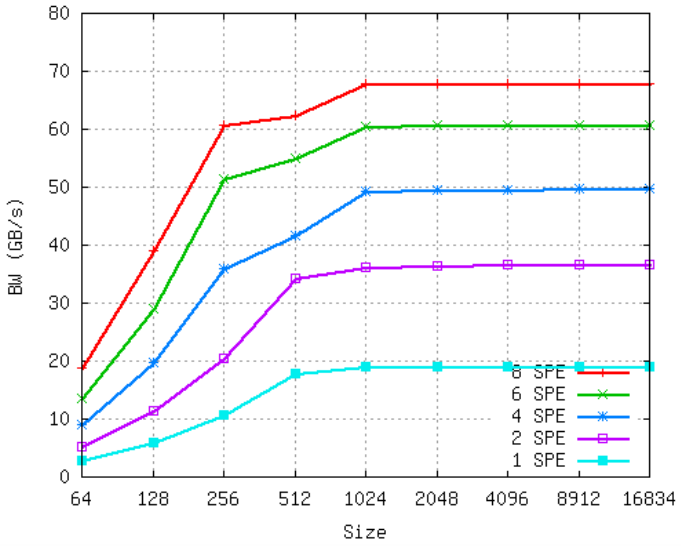
Interconnection Network BW vs. Size of transfers (SPE-SPE cycle)

Cellsim (4-Bus)

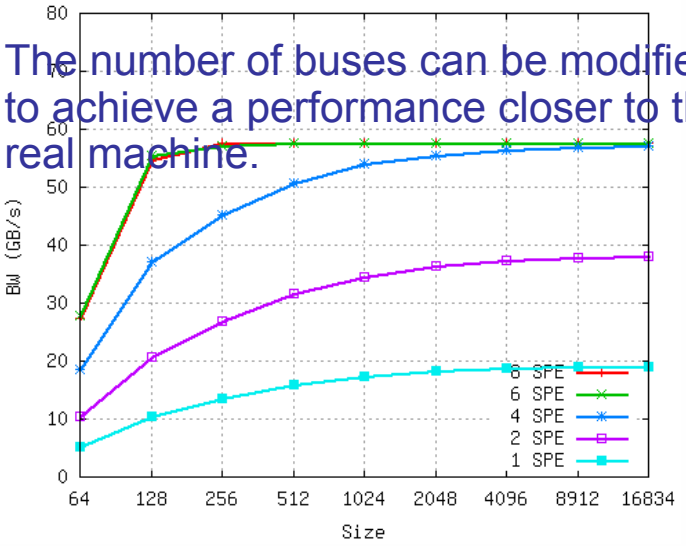


- In the experiments with 1 and 2 SPEs the peak bandwidth is reached by the Cell and the Cellsim.
- In Cellsim, for more than 4 SPEs, the bandwidth is limited by the number of buses.
- The Cell has lower bandwidth for 4, 6 and 8 SPEs because of contention due to SPE layout. This problem is not present in the Cellsim k-bus interconnection network.

Cell



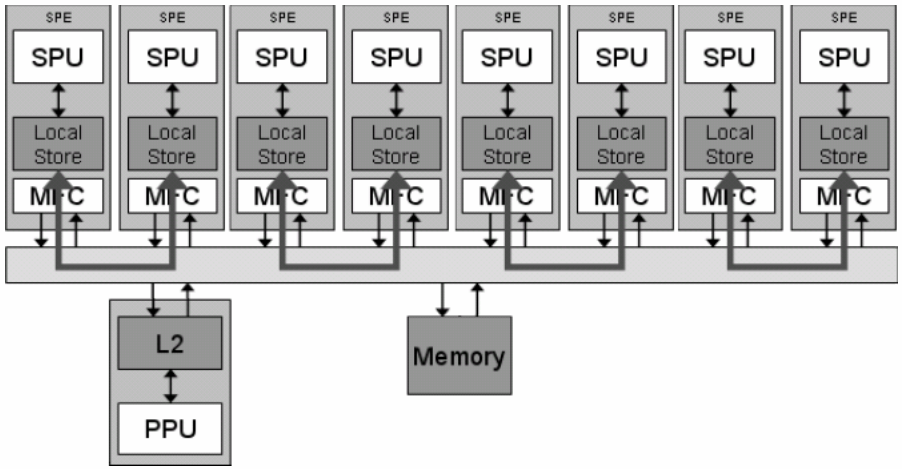
Cellsim (3-Bus)



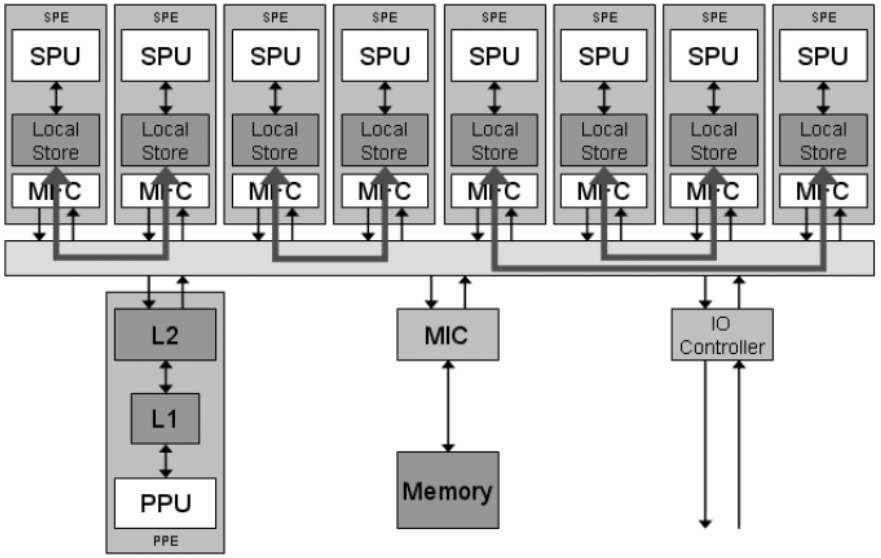
- The number of buses can be modified to achieve a performance closer to the real machine.

SPEs communication in couple layout validation program

- SPE-SPE couple: the SPEs perform DMA transfers distributed by couples
  - Each SPE sends and receives a total of 32MBytes of data with DMA operations of the following sizes: 64, 128, 256, 512, 1024, 2048, 4096, 8192 and 16384 bytes.



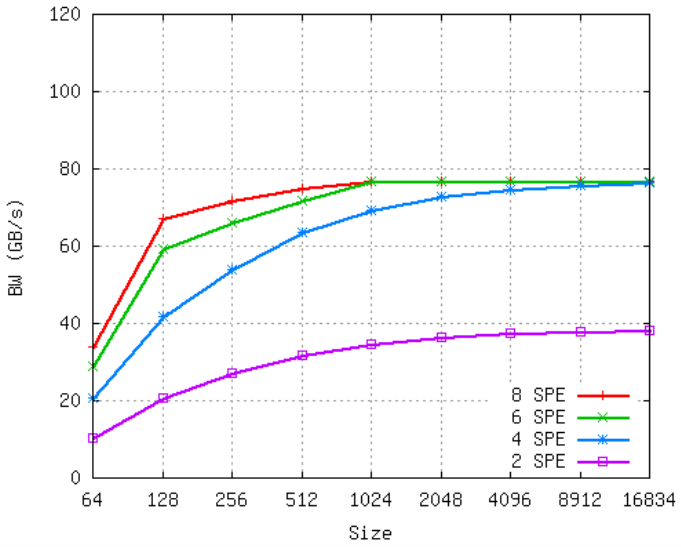
Cellsim Experiment



Cell Experiment

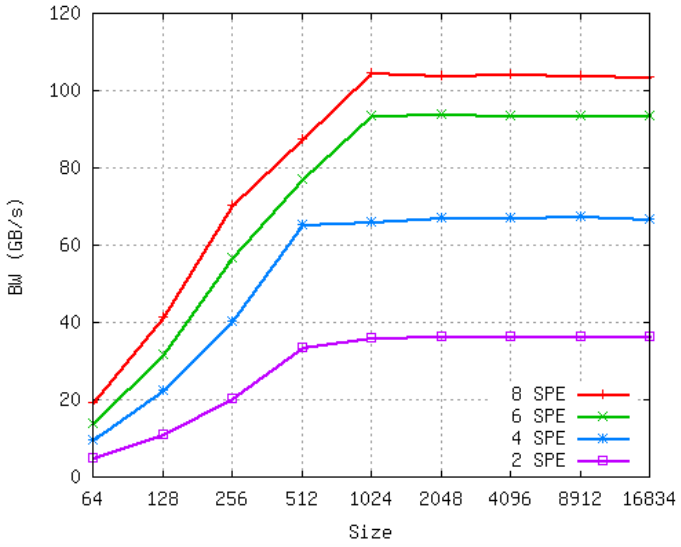
Interconnection Network BW vs. Size of transfers (SPE-SPE couple)

Cellsim (4-Bus)

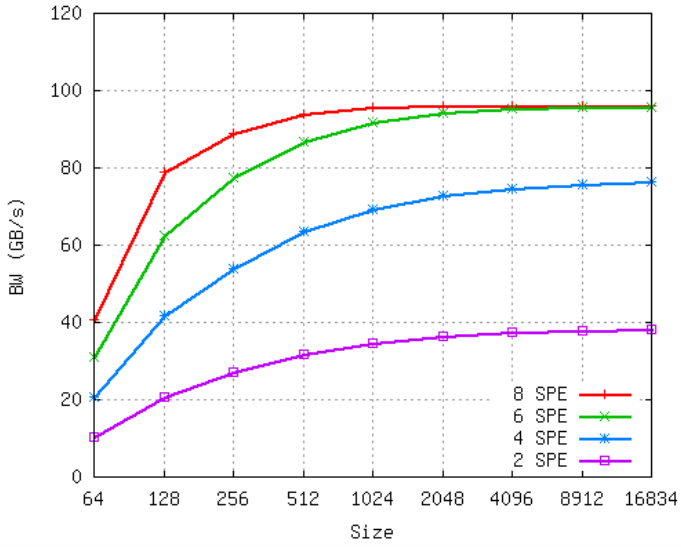


- The SPE-SPE couple gets a better SPE layout in the Cell that allows it to perform more than one transfer at a time per ring.
- This allows the Cell to get more bandwidth than the 4-bus interconnection network of the Cellsim.

Cell



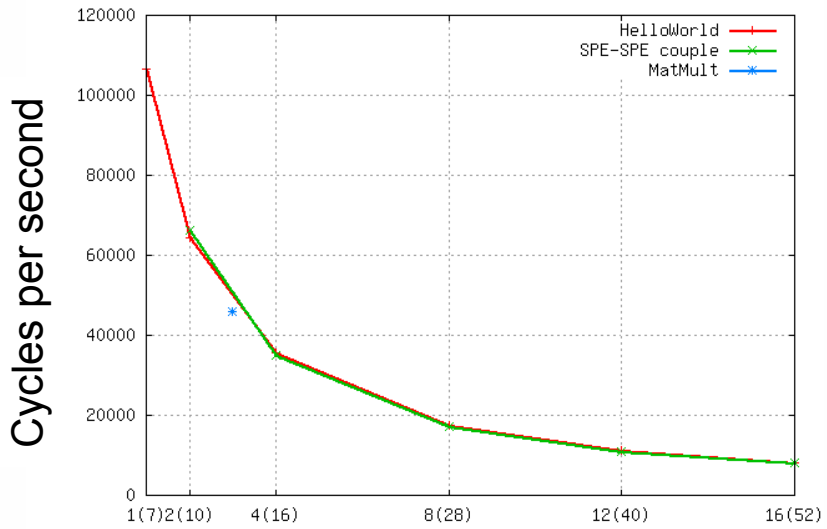
Cellsim (5-Bus)

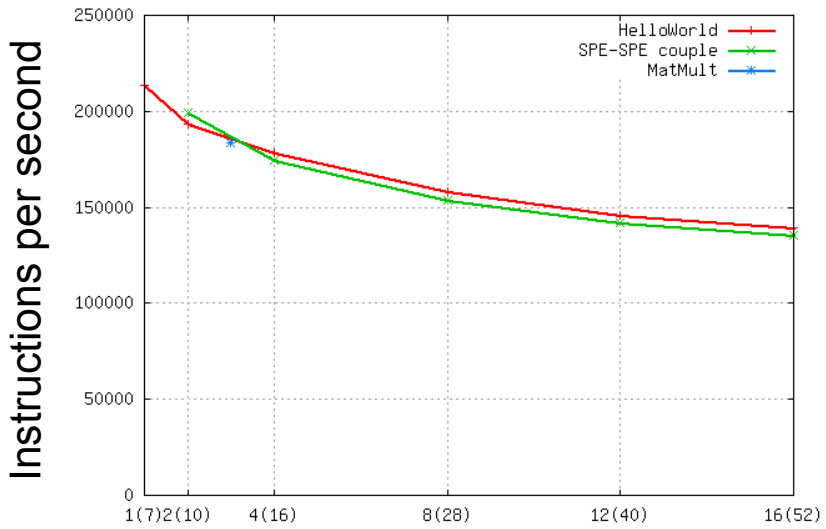


Validation Process Outline

- Cellsim Structure
- Modules description
- Validation
- **Cellsim performance**
  - **Speed**
  - **Profiling**

Speed





- The performance is not application dependent. There is a big potential for improvement in the infrastructure.

(If each PPU and SPU executes 1 instruction each cycle)

HelloWorld: each SPE executes a printf (not DMA commands)
SPE-SPE couple: continuous DMA traffic
MatMult: two SPEs perform a 64x64 elem matrix multiplication (one of them using intrinsics), a third compares the results (one DMA command to load data and another to store the result)
Configuration: 1 PPE, 4-bus interconnection network, issue width=1 for PPU and SPUs

Host Machine Configuration	
Processor	Intel Pentium 4 630@3.0GHz 2MB(L2), HyperThreading disabled
Memory	2 x 512 MB PC-4300(266MHz) DDR2-SDRAM (Dual Channel)
OS	Ubuntu Linux 6.06 i686

Profiling

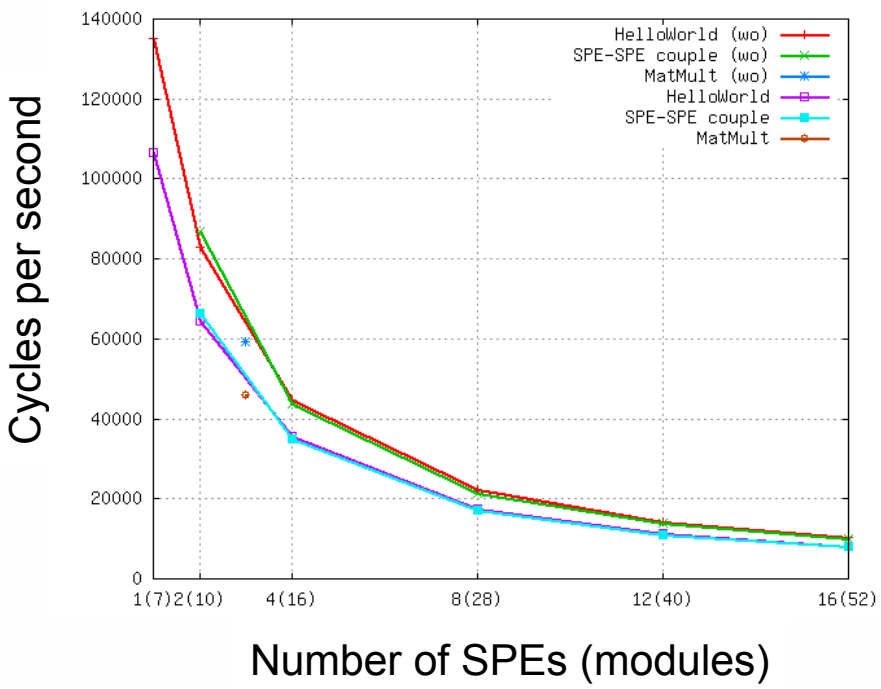
- Profiling of Matrix Multiplication program obtained using Pin [2].
  - Number of times each functions is called.
  - Number of instructions executed.

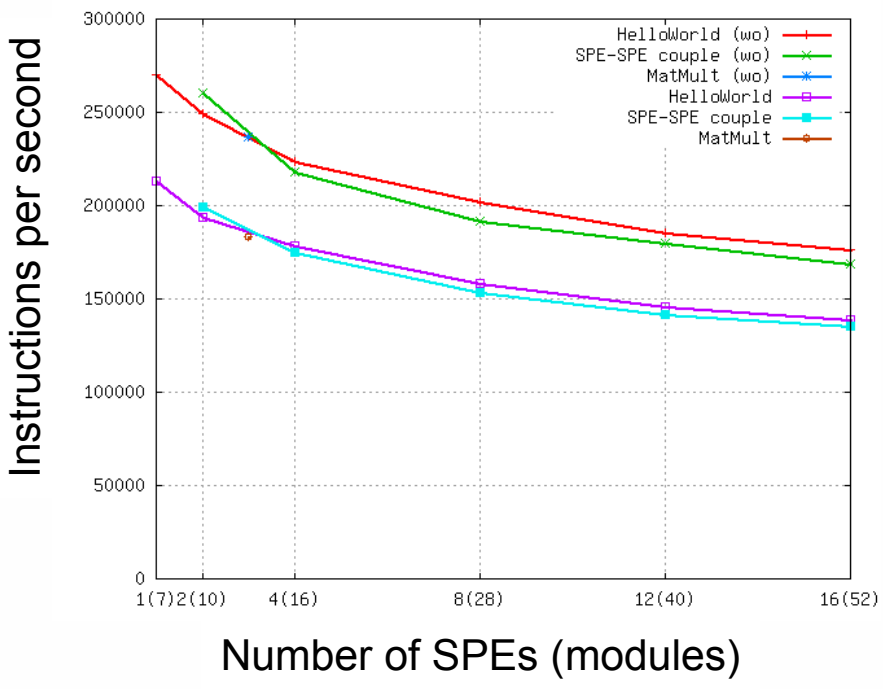
Procedure	Calls	Calls/Cycle	Instructions	%	Cumulative %
fsc_phase	9.8948E+06	2.00	2.2073E+10	18.25	18.25
outport<simulator::MemoryAccess>check_my_knowness	1.7811E+08	36.00	8.5491E+09	7.07	25.32
inport<simulator::MemoryAccess>check_my_knowness	1.7811E+08	36.00	8.5491E+09	7.07	32.39
unisim_port_check_knowness	4.9474E+06	1.00	6.4663E+09	5.35	37.74
ListPointer_unisim_port->	4.5516E+08	92.00	4.9969E+09	4.13	41.87
ListPointer_unisim_port=	4.4527E+08	90.00	4.8979E+09	4.05	45.92
ListPointer_unisim_port++	4.4527E+08	90.00	4.8979E+09	4.05	49.98
LS<3,262144,0>on_input	9.8948E+06	2.00	1.9436E+09	1.61	51.58
EIB<1,3>on_accept	9.8948E+06	2.00	1.8850E+09	1.56	53.14
LS<3,262144,1>on_input	9.8948E+06	2.00	1.8750E+09	1.55	54.69
LS<3,262144,2>on_input	9.8948E+06	2.00	1.8652E+09	1.54	56.23
EIB<1,3>end_of_cycle	4.9474E+06	1.00	1.6818E+09	1.39	57.63
LS<3,262144,0>on_enable	1.4246E+07	2.88	1.5586E+09	1.29	58.91
LS<3,262144,2>on_accept	9.8948E+06	2.00	1.4941E+09	1.24	60.15
LS<3,262144,1>on_accept	9.8948E+06	2.00	1.4936E+09	1.24	61.39
LS<3,262144,0>on_accept	9.8948E+06	2.00	1.4892E+09	1.23	62.62
outport<SPUStatus>check_my_knowness	2.9684E+07	6.00	1.4249E+09	1.18	63.80
inport<SPUStatus>check_my_knowness	2.9684E+07	6.00	1.4249E+09	1.18	64.97

Performance analysis

- At least 50% of the instructions executed belong to UNISIM communication procedures (Infrastructure).
- A large amount of the total instructions belong to procedures that check if all signals are set each cycle.
  - A 26% performance improvement is obtained on the simulations when we disable this feature.

Speed improvement





- This performance will be improved when Cellsim migrates to TLM UNISIM version.

(If each PPU and SPU executes 1 instruction each cycle)

Profiling

- There are still opportunities to improve the performance:
  - By migrating Cellsim to System-level UNISIM, all blue procedures should “disappear”.

Procedure	Calls	Calls/Cycle	Instructions	%	Cumulative %
fsc_phase	9.8948E+06	2.00	2.2073E+10	28.36	28.36
LS<3,262144,0>on_input	9.8948E+06	2.00	1.9436E+09	2.50	30.85
EIB<1,3>on_accept	9.8948E+06	2.00	1.8850E+09	2.42	33.27
LS<3,262144,1>on_input	9.8948E+06	2.00	1.8750E+09	2.41	35.68
LS<3,262144,2>on_input	9.8948E+06	2.00	1.8652E+09	2.40	38.08
EIB<1,3>end_of_cycle	4.9474E+06	1.00	1.6818E+09	2.16	40.24
LS<3,262144,0>on_enable	1.4246E+07	2.88	1.5586E+09	2.00	42.24
LS<3,262144,2>on_accept	9.8948E+06	2.00	1.4941E+09	1.92	44.16
LS<3,262144,1>on_accept	9.8948E+06	2.00	1.4936E+09	1.92	46.08
LS<3,262144,0>on_accept	9.8948E+06	2.00	1.4892E+09	1.91	47.99
EIB<1,3>on_data	4.9474E+06	1.00	1.4092E+09	1.81	49.80
EIB<1,3>start_of_cycle	4.9474E+06	1.00	1.1165E+09	1.43	51.24
LS<3,262144,1>on_enable	1.0277E+07	2.08	1.0693E+09	1.37	52.61
LS<3,262144,2>on_enable	9.8948E+06	2.00	1.0192E+09	1.31	53.92
SPU<1,0,0>on_fetch_enable	4.9474E+06	1.00	9.3040E+08	1.20	55.12
MFC<0>start_of_cycle	4.9474E+06	1.00	8.0671E+08	1.04	56.15
MFC<1>start_of_cycle	4.9474E+06	1.00	8.0660E+08	1.04	57.19
MFC<2>start_of_cycle	4.9474E+06	1.00	8.0643E+08	1.04	58.22

Summary

- We are developing a modular Cell simulator using UNISIM as the infrastructure.
- We have established a common interface for all modules to allow its reusability and interchangeability.
- The PPU and SPU are functional simulators.
- We are validating the PPU and SPU ISAs comparing the registers state with the real machine one.
- The interconnection network can be parameterized to achieve close performance to the Cell's one.
- There is a potential for Cellsim performance improvement in the infrastructure.

The End

- Thanks!