Case Study on Performance Tuning of Applications

Gadi Haber
Vadim Eisenberg
Marcel Zalmanovici
Source-level Performance Tuning

- A way for identifying and resolving logical & algorithmic issues in performance early in development stages
- Enables to address performance issues throughout the development life cycle
- Helps building an accurate performance modeling
- Performance tuning can start from Unit Test stage on
Source-level Techniques for Handling Performance Issues

- Code Reordering & Code re-partitioning
- Memoization
- Function Inlining & Specialization
- Tuning Hot loops using history information
- Field Packing
- Software Caching
Identifying Performance Issues

When using representative input workload, logical profiling of the code can identify:

- Hot vs. Cold code
- Hot loops
- Hot memory references of load instructions
- Hot I/O operations
Code Reordering & Re-partitioning

- **Separating Hot code from Cold code** - done by relocating
  - rarely executed segments of code
  - error-handling modules
- **Turning Hot code segments into functions** - for hot segments that are called from multiple places
- **Performing memoization** - for computational functions or code segments
- **Function inlining & Specialization** - for functions that are called from a single dominant site
Hot loops are characterized by:

- performing a large number of iterations
- being called a large number of times
3 Important cases of Hot loops:

1. loops performing a certain computation
   - Maintain temporal computational results for future loops' invocations
2. loops performing heavy I/O operations
   - Map I/O into memory (when possible)
3. loops searching in or traversing thru data structures
   - Maintain pointers to frequently referenced elements to be used for future searches of the loop
Hot Loops - Bounded Behavior Example

![Graph showing iteration number vs. loop call count](image-url)
Hot Loops- Linear Behavior Example
Two possible cases of bottlenecks

1. Distances between referenced addresses are large
2. The references pattern is irregular
How to handle data $\$ bottlenecks

- When the average stride between referenced memory addresses is large, i.e., the addresses are located far from one another
  - Perform Field Packing
- When no average stride exists, i.e., the instruction "jumps" irregularly between memory locations
  - Perform software caching or replace the allocation algorithm of the referenced data structure