Application Awareness in Adaptation Middleware: Balancing Transparency with Performance and Adaptivity


Queen's University Belfast - Research Portal:
Link to publication record in Queen's University Belfast Research Portal

General rights
Copyright for the publications made accessible via the Queen's University Belfast Research Portal is retained by the author(s) and / or other copyright owners and it is a condition of accessing these publications that users recognise and abide by the legal requirements associated with these rights.

Take down policy
The Research Portal is Queen's institutional repository that provides access to Queen's research output. Every effort has been made to ensure that content in the Research Portal does not infringe any person's rights, or applicable UK laws. If you discover content in the Research Portal that you believe breaches copyright or violates any law, please contact openaccess@qub.ac.uk.
Application Awareness in Adaptation Middleware: Balancing Transparency with Performance and Adaptivity

Dimitris Nikolopoulos
Computer Science
College of William&Mary
Axes of adaptation

- Execution
- Environment
- Architecture
- Algorithm
Transparent Adaptation

• Ideally:
  – Perfect knowledge of hardware
    • compiler, or exhaustively tuned libraries
  – Perfect knowledge of execution conditions (OS)
  – Instantaneous reaction to changes (algorithm)

• In reality
  – Hardware adaptation possible
  – Adaptation to execution conditions difficult
    • for both the algorithm and the system interface
    • Lack of information, slow reaction time
Experiences

• Two “simple” forms of adaptation
  – Adaptation to varying NCPUUS (shared-memory)
  – Adaptation to varying memory/cache space

• Nanotreads projects (UIUC Promis compiler, NANOS compiler and RTL)
  – Targeted OpenMP and SMPs (ccNUMA )

• Adaptive Memory Management Library (WM)
  – Targets non-dedicated COWs
Adaptation through scheduling

• Poor OS/application interaction
  – Do I have one more processor to use?
  – Did I just lose one processor?
  – Am I preempted in a critical section/on the critical path?

• Solution:
  – A lightweight interface (polling with loads and stores)
  – Critical information exchange
  – Transparent application adaptation through scheduling in the RTS
  – Constraint: utilized better by a 2-L thread model
  – Constraint #2: May need OS extensions
The Shared Arena

Application User Space

R/W segment

- n_cpu_requested
- Worker/idler
- blocked/preempted/running

R/O segment

- n_cpu_current
- n_cpu_preempted

Kernel Space

- adaptive parallel programs
- OS Scheduler

- non adaptive parallel and sequential programs

Dimitrios S. Nikolopoulos
Using the shared arena

• Polling/write interface (encapsulated in the RTS)
  – Well-defined polling points
    • Thread create
    • Dependence trigger
    • Synchronization (lock/barrier)

• A convenient recovery mechanism
  – If preempted and more critical than some running thread
    recover ASAP

• Transparency:
  – Invisible to the application, only information needed is
    worker/idler
User-level thread priorities

• Assess thread criticality within the application

• Criticality criteria:
  – Synchronization execution status
  – Location on dependence path

• Two-level scheduling
  – CPU time to the application instead of threads
  – Let the application to the TDM
Locality adaptation

- Changing NCPUS implies migrations
- Migrations move threads away from data
- Competitive argument
  - move data to threads
  - move threads to data
- Developed UPMlib (ICS’ 2000) to handle this issue
  - transparently detect migrations
  - use hardware counters to infer affinity
  - move data close to migrated threads, with a competitive argument
Putting it all together

• NANOS library (ICS’ 98, PDCS’ 99, JSSPP’ 00)
  – An adaptive RTS for OpenMP
  – Transparent adaptation to multiprogramming
  – Dynamic thread control

• SyncLib (ICPP’ 01)
  – optimized synchronization with inner priorities

• UPMlib (ICS’ 00, LCR’ 00, SC’ 00, ICPP’ 00, JPDC’ 02, SP’ 00)
  – coordinated movement of threads and data
Results (64-CPU Origin2000)

**Execution Time**

- **Optimal**
- **IRIX**
- **Adaptive, Sync**
- **Adaptive, Sync, Locality**

**Graph Details:***

- The x-axis represents different configurations: Optimal, IRIX, Adaptive, Sync, Adaptive, Sync, Locality.
- The y-axis represents execution time in seconds, ranging from 0 to 350.
- The graph compares two benchmarks: NAS BT and NAS SP.

**Legend:**

- **NAS BT** (Green)
- **NAS SP** (Blue)

2/13/12 SIAM PP’ 04

Dimitrios S. Nikolopoulos
Adaptive Memory Management

• Motivation
  – Harness cycles with data-intensive apps in non-dedicated, non batch-scheduled COWs
  – Execute with variable amounts of available memory
  – …without the OS VMM problems

• Application does not really know
  – How many pages are in/out ?
  – What is the actual replacement policy ?
  – Which page will be evicted next ?
Two views

• RTS/OS view (IPDPS’ 2002, CCGrid’ 2002)
  – Use shared arena for co-scheduling and MM
  – Can control size of memory footprint with a system-specific unit (e.g. page)
  – Can detect availability, shortage
  – Can not control replacement policy
  – Substantial OS extensions

• Application/RTS view (IPDPS’ 2003, IPDPS’ 2004)
  – Can control size of memory footprint with application-specific unit (e.g. panel)
  – Can detect availability (like RTS)
  – Can customize reclaim/eviction policy
Portable framework for memory adaptivity

• Looking for a pure user-level solution
• Exploit panel structure
• Panels are
  – An application-dependent indivisible work unit
  – A unit of adaptivity at all levels of the memory hierarchy
• Basic idea
  – Seamless switch between in-core and out-of-core
  – React to memory surplus and shortage with panel reclamation and eviction
Challenges

- In-core must be as fast as non-adaptive version
- Both memory availability and memory shortage need to be detected
  - OS’es do not provide this information
- Application-dependent algorithms for replacing/reclaiming blocks
  - Replacement algorithm per object
  - Differentiate between static and dynamic memory
Solutions

- Memory-mapped I/O
  - With full availability match in-core performance
  - Control the swap space locally
- Control on a panel-by-panel basis
  - Whenever a new panel is brought in to work, grow, shrink, or keep working
- Application-specific policy for panel eviction
  - e.g MRU for repeating sequential access pattern
  - different eviction policy per object
Detecting memory shortage/availability

- One variable - RSS
  - Reported accurately from OS

- Memory shortage
  - Compare current RSS with RSS during last panel
  - If RSS drops memory shortage

- Memory surplus
  - No OS mechanism, use probing
  - Compare RSS to “desired” RSS, grow if RSS >= dRSS
  - Use delayed growth
    \[
    \frac{\max RSS}{\max RSS - RSS}
    \]
Results (no probing optimizations)
Results (controlled probing delay)

RSS (solid line) and desired RSS (dashed line) for dynamic delay with ratio

Average iteration time: 4.4 sec
Results (graceful degradation)

Average time per matrix-vector multiplication vs. size of competing job

Average time per iteration (seconds)

Size of competing job (MB)
Baseline performance

• Modified Gram-Schmidt
  – Dedicated in-core 9.1s
  – Dedicated adaptive 9.9s
  – Small memory pressure (140 MB on 128 MB machine)
  – Multiprogram in-core 172 seconds
  – Multiprogram adaptive 39 seconds

• Ising model:
  – Slight memory pressure (1-2 MB)
  – Dedicated in-core 43.2 seconds
  – Dedicated adaptive 46.3 seconds
  – Multiprogram in-core 350+ seconds
  – Multiprogram adaptive 95 seconds
Conclusions

• Adaptation requires interoperation between
  – Application
  – Runtime system
  – Operating system
  – Difficult to decide what goes where

• Transparency through runtime libraries
  – Lightweight interfaces, preferably user-level
  – Generalized abstractions that can be downloaded to libraries (e.g. panels, threads)
  – Plug-in policies
  – Can everything be done through adaptive libraries?
Acknowledgements

• Nanothreads and related projects
  – Christos Antonopoulos (Patras), Xavier Martorell, Julita Corbalan, Jesus Labarta (UPC)
    Constantine Polychronopoulos (UIUC)

• Adaptive execution on COWs
  – Andreas Stathopoulos, Richard Mills (W&M)

• Funding
  – EU LTR No. 21907, W&M, NSF ITR
    ACS-0082094, ITR AP-0112727, DOE Graduate Fellowship