Why you should care about hardware locality and how.

Brice Goglin – TADaaM team – Inria Bordeaux Sud-Ouest
Agenda

- Quick example as an introduction
- Bind your processes
- What's the actual problem?
- Convenient modeling and management of the architecture with hwloc
- So, where do I place my tasks?
- Conclusion
Quick example as an Introduction
Machines are increasingly complex
Machines are increasingly complex

- Multiple processors
- Multicore processors
- Simultaneous multithreading
- Shared caches
- NUMA
- Multiple GPUs, NICs, ...

- We cannot expect users to understand all this
Example with MPI

- PlaFRIM (Bordeaux) is installing is new cluster
  - 12-core Xeon E5-2600v3 with NVIDIA K40, etc.

- Nice, let's run some benchmarks!
  - Open MPI 1.8.1
  - Intel MPI benchmarks 3.2
Example with MPI (2/3)

- Between cores 0 and 1
  - 540ns, 3500MiB/s
- Between cores 0 and 2
  - 330ns, 4220MiB/s
- Between cores 0 and 12
  - 430ns, 4290MiB/s
- Between cores 0 and 23
  - 590ns, 3410MiB/s
What is going on?
Example with MPI (3/3)

- Between cores in same NUMA node
  - 330ns, 4220MiB/s
- Between cores in different NUMA nodes of same processor
  - 430ns, 4290MiB/s
- Between cores in different processors
  - 540ns, 3500MiB/s
- Between cores in different processors and NUMA nodes far away from each other
  - 590ns, 3410MiB/s
What about AMD machines? 
Even worse!
First take away messages

- Locality matters to communication performance
  - Machines are really far from flat

- Cores numbering is crazy
  - Never expect anything sane
It's actually worse than that

GPUs attached to one NUMA node
I/O affinity

- If you use GPUs or high performance networks, you must allocate host memory close to them
  - Otherwise DMA to GPUs slows down by 10-20% here
  - InfiniBand latency increases by 10%

- Need a way to know which cores/memory is close to which I/O device
Bind your processes
Where does locality actually matter?

- MPI communication between processes on the same node
- Shared memory too (threads, OpenMP, etc.)
  - Synchronization
    - Barriers use caches and memory too
  - Concurrent access to shared buffers
    - Producer-consumer, etc.
- 15 years ago, locality was mostly an issue for large NUMA SMP machines (SGI, etc.)
  - Today it's everywhere
    - Because multicores and NUMA are everywhere
What to do about locality in runtimes?

- Place processes/tasks according to their affinities
  - If two tasks communicate/synchronize/share a lot, keep them physically close
    - Main focus of this talk
- Adapt your algorithms to the locality
  - Adapt communication/synchronization implementations to the topology
    - Ex: hierarchical OpenMP barriers
    - Another example in the next slide
Adapting MPI implementation thresholds to shared caches

Threshold between strategies
Depends on cache size, contention, etc.
Process binding

- Some MPI implementations bind processes by default (Intel MPI, Open MPI 1.8)
  - Because it's better for reproducibility
- Some don't
  - Because it may hurt your application
    - Oversubscribing? Dynamic processes?
- Binding doesn't guarantee that your processes are optimally placed
  - It just means your processes won't move
    - No migration, less cache issues, etc.
To bind or not to bind?

Zeus MHD Blast. 64 Processes/Cores. Mvapich2 1.8. + ICC
Ok, I need to bind.
But where?

- Default binding strategies?
  - By core first (compact, --map-by core, etc.)
    - One process per core on first node, then one process per core on second node, ...
  - By node first (scatter, --map-by node/socket, etc.)
    - One process on first core of each node, then one process on second core of each node, ...

- Your application likely prefers one to the other
  - Often the first one
    - Because your algorithms often communicate more between immediate neighbors
  - Sometimes the other one...
Binding strategy impact

![Graph showing the impact of different binding strategies on execution time](image)

- No binding
- Binding by Node first
- Binding by Core first
How do I choose?

• Dilemma
  • Use cores 0 & 1 to share cache and improve synchronization cost?
  • Use cores 0 & 2 to maximize memory bandwidth?
• Depends on the application needs
  • And machine characteristics
• More about this later
3 What's the actual problem?
Example of dual-Xeon machine
Another similar machine?
Processor and core numbers are crazy

- Resource ordering/numbering is unpredictable
  - Ordering by any combination of NUMA/processor/core/hyperthread
  - Can (and does) change with the vendor, BIOS version, etc.
- Some resources may be unavailable
  - Batch schedulers allocates parts of machines
    - Core numbers may be non-consecutive, not start at 0, etc.
- Don't assume anything about these numbers
  - Otherwise your code won't be portable
Vertical ordering of levels (who contains who)
Vertical ordering isn't reliable either

- Modern processors (Xeon E5v3, Opteron 6000, Power8) have 2 NUMA nodes each
  - Old platforms have multiple processor packages per NUMA nodes
- Levels of caches and sharing may vary

- Don't assume anything about vertical ordering
  - Or (again) your code won't be portable
  - e.g.: Even the Intel OpenMP binding isn't always correct
Gathering topology information is difficult

- Lack of generic, uniform interface
  - Operating system specific
    - /proc and /sys on Linux
    - rset, sysctl, lgrp, kstat on other OS
  - Hardware specific
    - x86 CPUID instruction, device-tree, PCI config space, etc.
- Evolving technology
  - AMD Bulldozer introduced dual-core Compute Units
    - It's not two real cores, neither one hyper-threaded core
    - New kinds of hierarchy/resources?
- And some BIOS report buggy information
Binding is difficult too

- Lack of generic, uniform interface (again)
  - Process/thread binding
    - `sched_affinity()` system call changed twice in Linux
  - Memory binding
    - 3 different system-calls on Linux
      - `mbind()`, `migrate_pages()`, `move_pages()`
  - Different constraints
    - Bind to single core only? To contiguous set of cores? To random sets of cores?
  - Many different policies
Convenient modeling and management of the architecture with hwloc
What do we need?

- Find neighbor cores
- Find cores that share a cache
  - Type and size of the cache
- Find cores near a memory bank
- Find the cores or memory banks near a GPU
What hwloc (Hardware Locality) is

• Detection of hardware resources
  • Everything that can run a task
    • And things that contain them: cores, packages, etc.
  • Memory nodes, caches
  • I/O devices

• Organized as a tree
  • Logical resources identification based on locality
    • No need to deal with crazy numbering anymore
What hwloc is (2/2)

- API and tools to consult the topology
  - Answer to all your needs
    - Thanks to the locality-based tree
    - Can be browsed in vertical (find container or contained resources) or horizontal manner (find neighbors)
  - Without caring of portability issues or crazy numbers anymore
- A portable binding API
  - Execution and/or memory binding to resources chosen in the tree
What hwloc is NOT

- A placement algorithm
  - hwloc only gives hardware information
  - You're the one that knows what your application does/needs
  - You're the one that can match software affinities to hardware locality
- A profiling tool
  - hwloc can match profilers' output with the topology to help finding bottlenecks
So, where do I place my tasks?
Topology isn't flat
Communication times aren't uniform
Communication patterns aren't flat either

- Some tasks communicate a lot with each other
  - The physical distance will slow down some messages
  - Try to keep them close!
- Some don't
  - No constraint on placement
Reordering tasks to improve locality

No binding communication pattern – ZeusMP/2
Metric : msg

TreeMatch communication pattern – ZeusMP/2
Metric : msg
Reordering with TreeMatch

- At process launch-time
  - mpiexec options
- Dynamically
  - MPI_Dist_graph_create() to swap MPI ranks' roles between application steps
  - Charm++ load-balancer

- The communication volume is unchanged
  - But big volumes move inside nodes
- Faster execution!
Communication pattern discovery

- TreeMatch uses a matrix of communication between all pairs of processes
  - Either volume or number of messages

- Given by the application developer?
- Static analysis of the code?
- Additional run for instrumentation
- Guessed from data partitioning
What about other programming languages?

- OpenMP
  - Communication pattern becomes buffer sharing and synchronization
  - The parallelism is often flat
    - Nested parallel loops poorly supported and rarely used
  - OpenMP tasks may be too dynamic?
    - Rather use StarPU-like task-graph models?
      - Locality managed through prediction models for tasks execution and data transfer times

- PGAS
  - Similar to MPI, easy?!
Better modeling for better decision?

- hwloc model doesn't say anything about contention
  - If 4 cores shared a cache or NUMA node, will they slow down if I run 4 processes there?

- Do we need very accurate performance modeling?
  - StarPU doesn't require very accurate prediction of execution times
  - TreeMatch doesn't require communication latency/bandwidth, logical distance is enough?
Don't expect accurate modeling of memory access performance

Store Hit performance on Xeon E5 depending on local cache state
6 Conclusion
Summary

- Machines aren't flat anymore
  - Ignoring this fact limits your application performance
    - And it's getting worse with new platforms
- Understanding how resources are organized is hard
  - We expose this information in abstracted and portable manner
- You can use it in your application to improve performance thanks to locality
  - Better placement of tasks
  - Map your algorithms on the hardware tree
How do I use hwloc?

- Distributed under the BSD license
- Already used by all MPI implementations, many batch schedulers, parallel libraries, etc.
- Packages available for most Linux distributions
- Building from source is easy

- lstopo and hwloc-* command-line tools
- But the actual power in the C API
- Perl and Python bindings
- Plenty of documentation online
Using TreeMatch

- Distributed under BSD license

- Included in upcoming Open MPI 1.9
  - Communication pattern discovery
  - Process reordering
Future work

- Network topology
  - Netloc component in hwloc
  - We cannot represent the topology as a tree anymore
    - Harder problems for algorithms
- Better modeling
  - Hints about contention, etc.
- Application modeling
  - More than a communication pattern
  - Hints about application needs
Thank you

http://team.inria.fr/tadaam
http://www.open-mpi.org/projects/hwloc
http://treematch.gforge.inria.fr

and thanks to my colleagues
B. Putigny, F. Tessier, G. Mercier & E. Jeannot

Brice.Goglin@inria.fr