Improving Performance Through Task Locality

Nicolas Denoyelle, Brice Goglin,
Jens Gustedt, Emmanuel Jeannot

Inria / ENSEIRB-MATMECA / Université de Bordeaux

Runtime Team

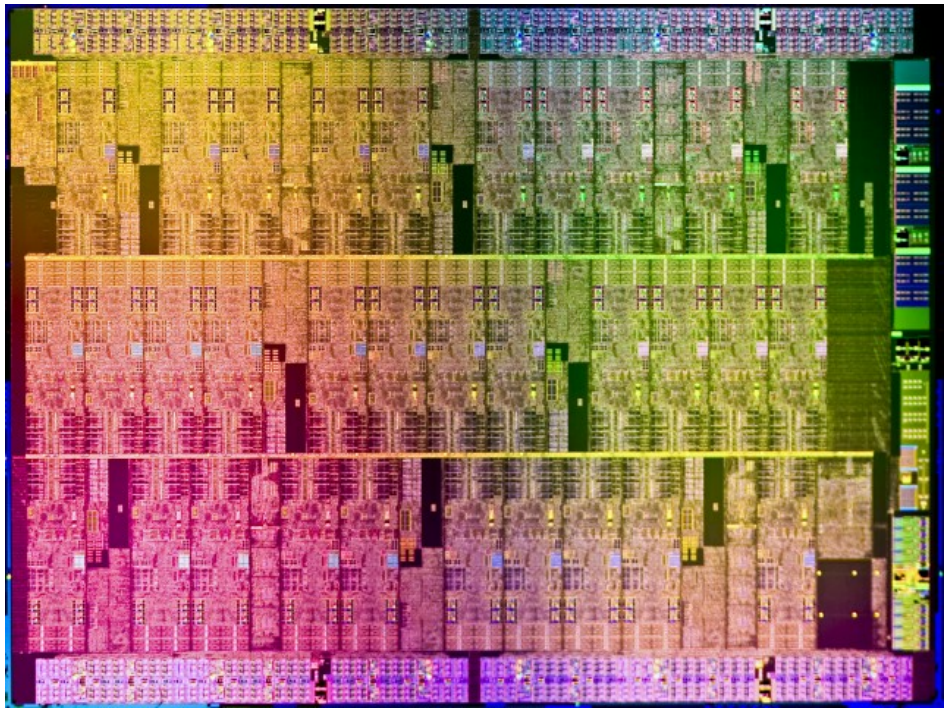
March 16, 2014

Overview

Core Density Growth

- Increasing number of Cores per chipset
- Decreasing memory per Core

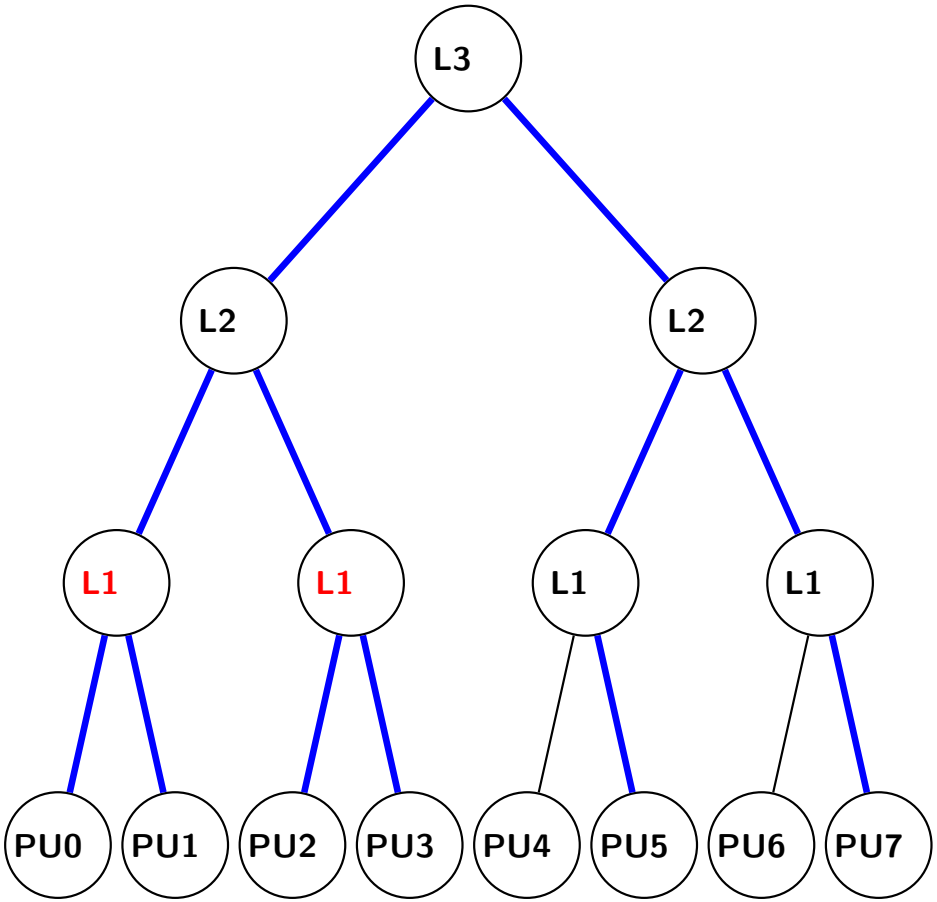
Memory access is a critical bottleneck.
Managing resources (device, Core, NUMA node) use improves data locality and performance.

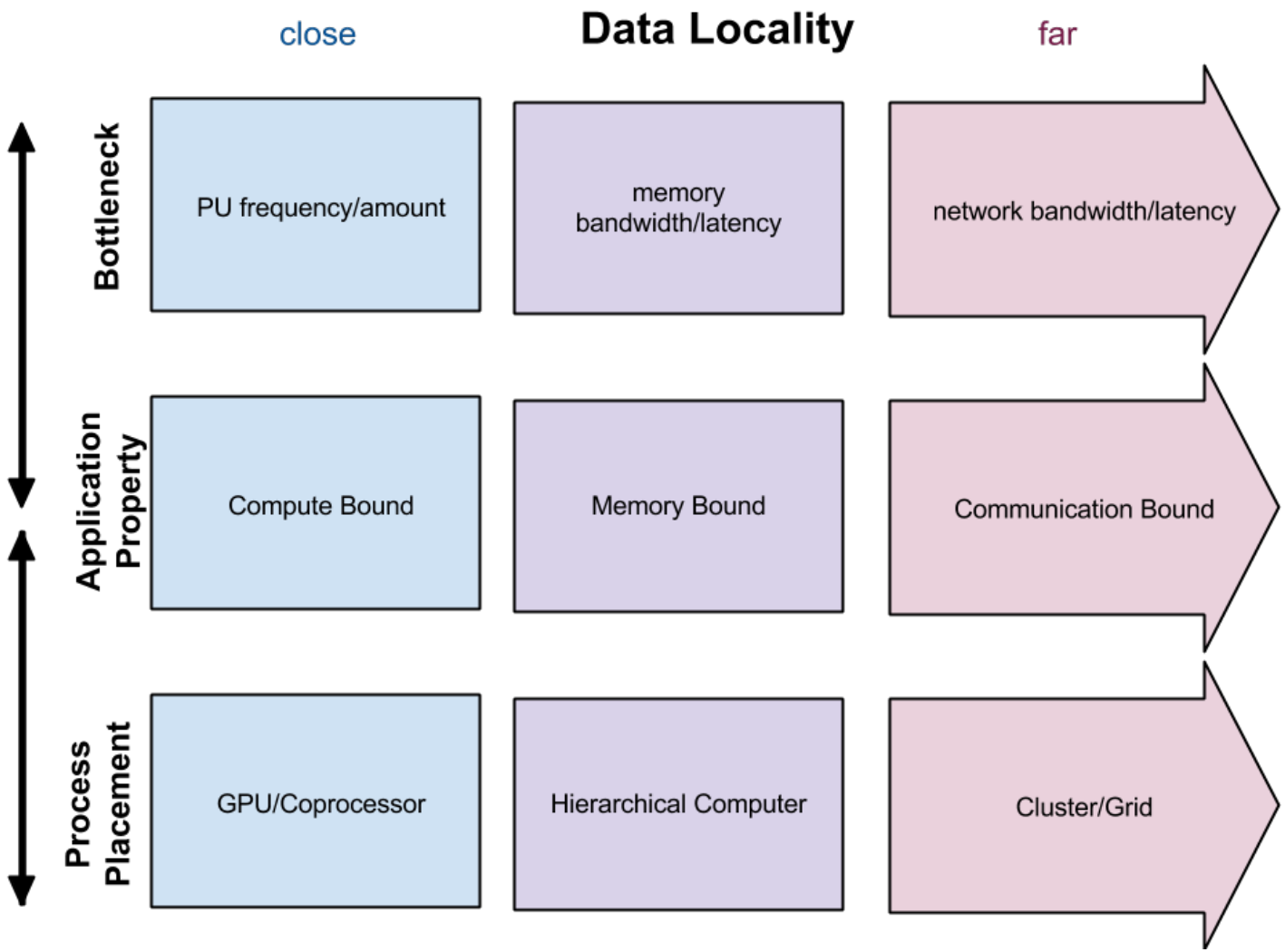


dot product

for $i \leftarrow 0..n$
$result \leftarrow result + array_A[i] * array_B[i]$

- Memory bound:
  n multiplications, $log(n)$ + additions
  $2 * n$ read operations
- L1 cache conflicts (left side) might be a bottleneck
- It may be better to use a single core under a memory node (L1 here).




close
Data Locality
far
Bottleneck
PU frequency/amount
memory bandwidth/latency
network bandwidth/latency
Application Property
Compute Bound
Memory Bound
Communication Bound
Process Placement
GPU/Coprocessor
Hierarchical Computer
Cluster/Grid

Given

- A set of task
- The hardware topology

Map tasks to hardware resources.

Objectives

- Identify bottlenecks in applications.
- Find efficient pattern of processus placement according to bottlenecks.
- Map applications tasks to resources according to those patterns.

State of Art

Placement Strategies

- TreeMatch[5]: Process placement based on topology and communication pattern
- Scotch,Metis[7]: Graph partitioners
- Charm++[6]: Dynamic load balancing runtime
- StarPU[1], Xkaapi[4]: Task oriented API & Runtime, to optimize resources use.
- and maaaaany others ...

→ Balance between explicit instructions and transparency.

hwloc[2]

hardware locality

- Portable API
- Gather and walk the system hierarchical topology
- Bind threads to CPU's and memory components

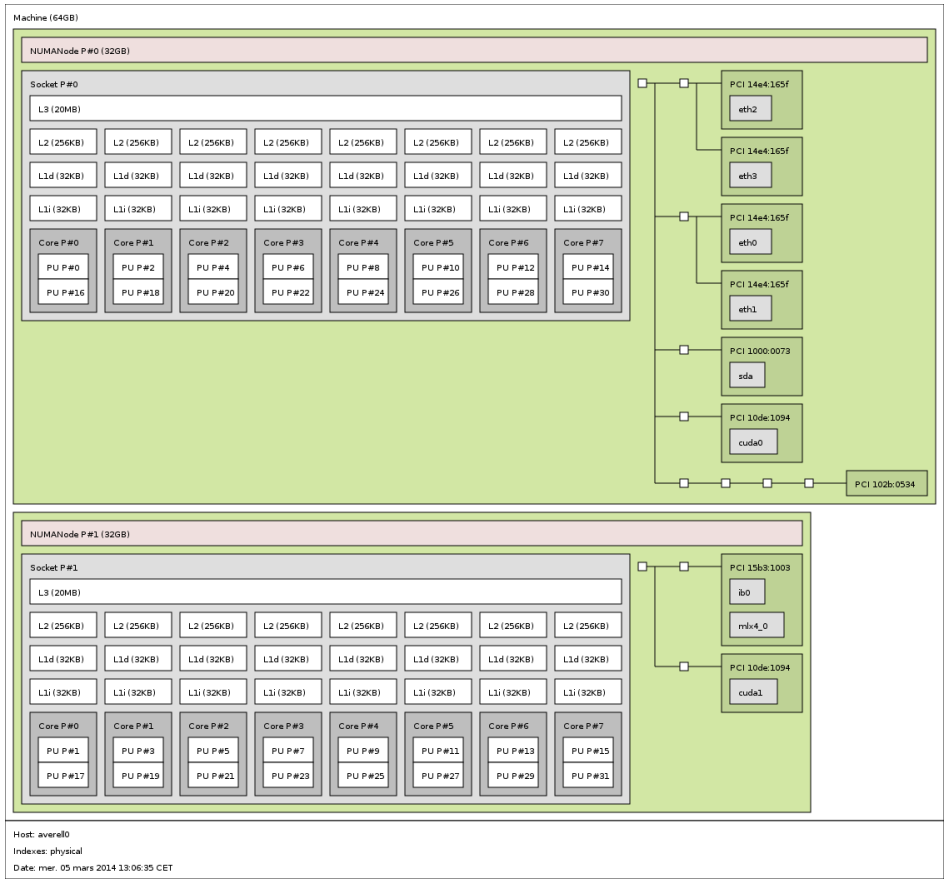


Figure : lstopo on an Intel Xeon E5-2650

ORWL[3]

Ordered Read Write Lock

- Framework: (API + Runtime)
- Resource access management
- Task programming paradigme
- Low overhead

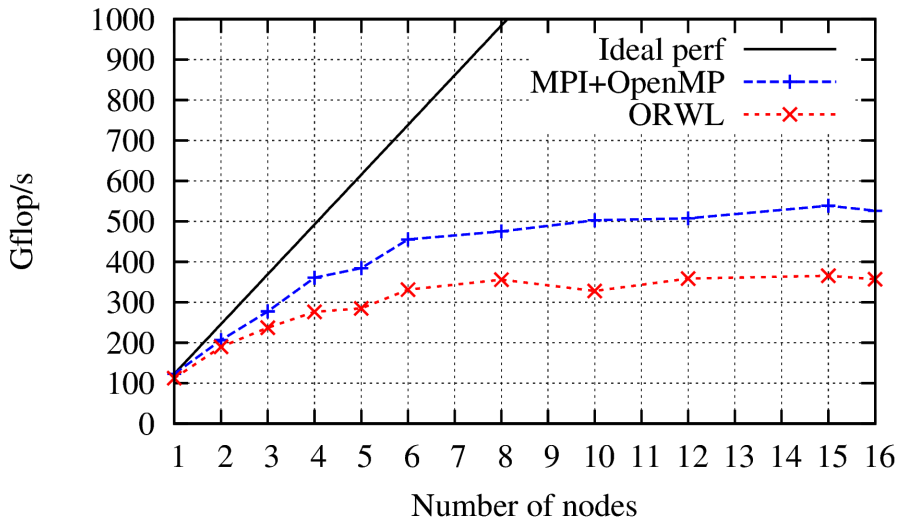


Figure : Gflop/s achieved on a constant size problem ($5670 * 5670$ matrices of double precision data) using only the CPU cores of *Cameron* cluster.

BLAS, LAPACK

- Exploiting different kind of bottlenecks
- Widespread benchmark
- Optimized library used in many scientific applications
- Multiple implementations

NAS Parallel Benchmark

- Effects on communication-bound applications
- Both contiguous and random memory access
- Benchmarks derived from computational fluid dynamics
- Needs to be reimplemented with ORWL

Measuring Time

- ORWL builtin measure tools

3 use cases

- compute-bound (Blas 3)
- memory-bound (Blas 1)
- communication-bound

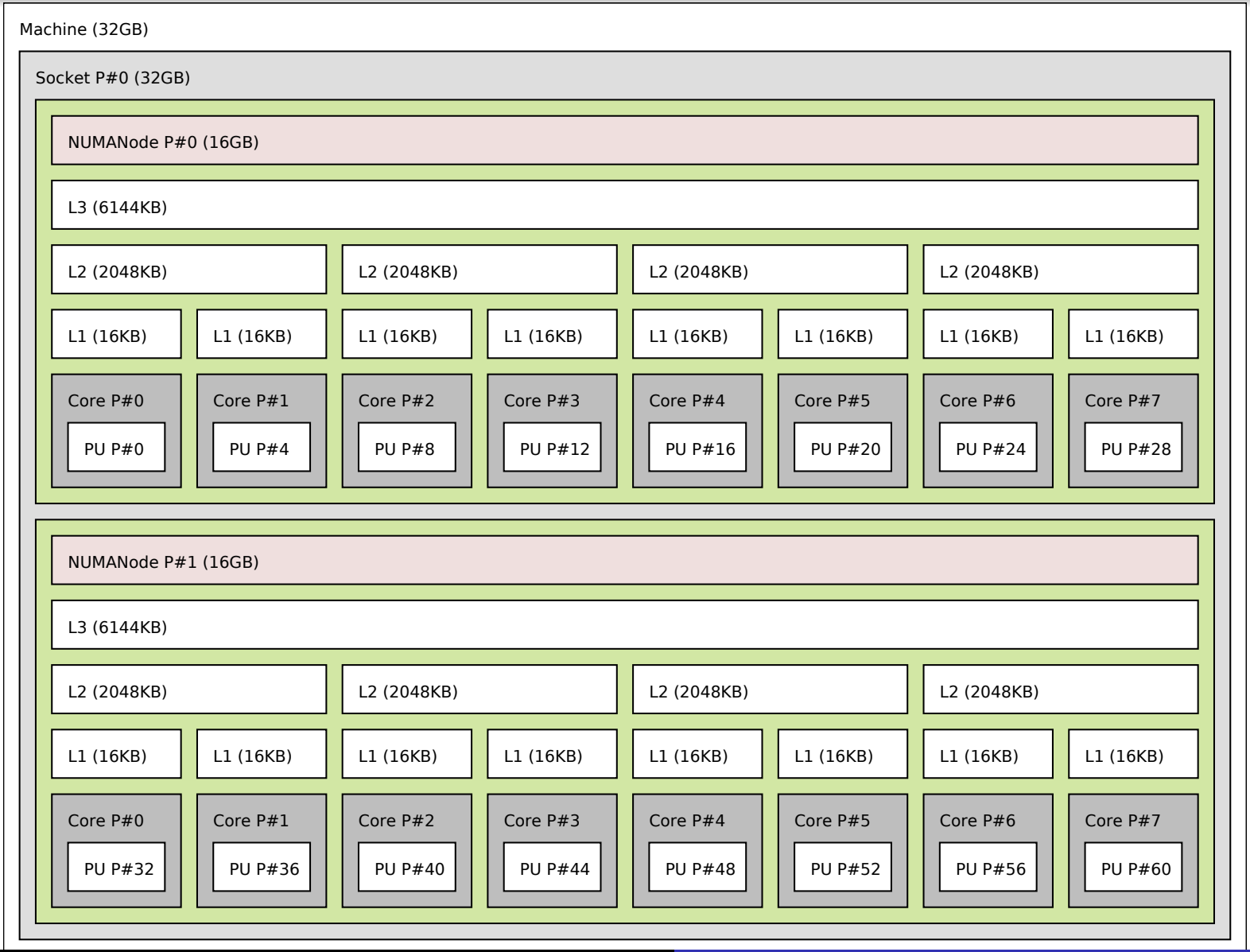
Requires a fine separation in tasks

Multiple resource (core, NUMA, . . .) ordering.

- One thread per task
- Several task per processing unit is better to avoid time loss while waiting for locks.

Bound

- Communication: TreeMatch + Bind near NIC.
- Compute: Bind near the same memory node.
- Memory: Bind single PU above a big cache.

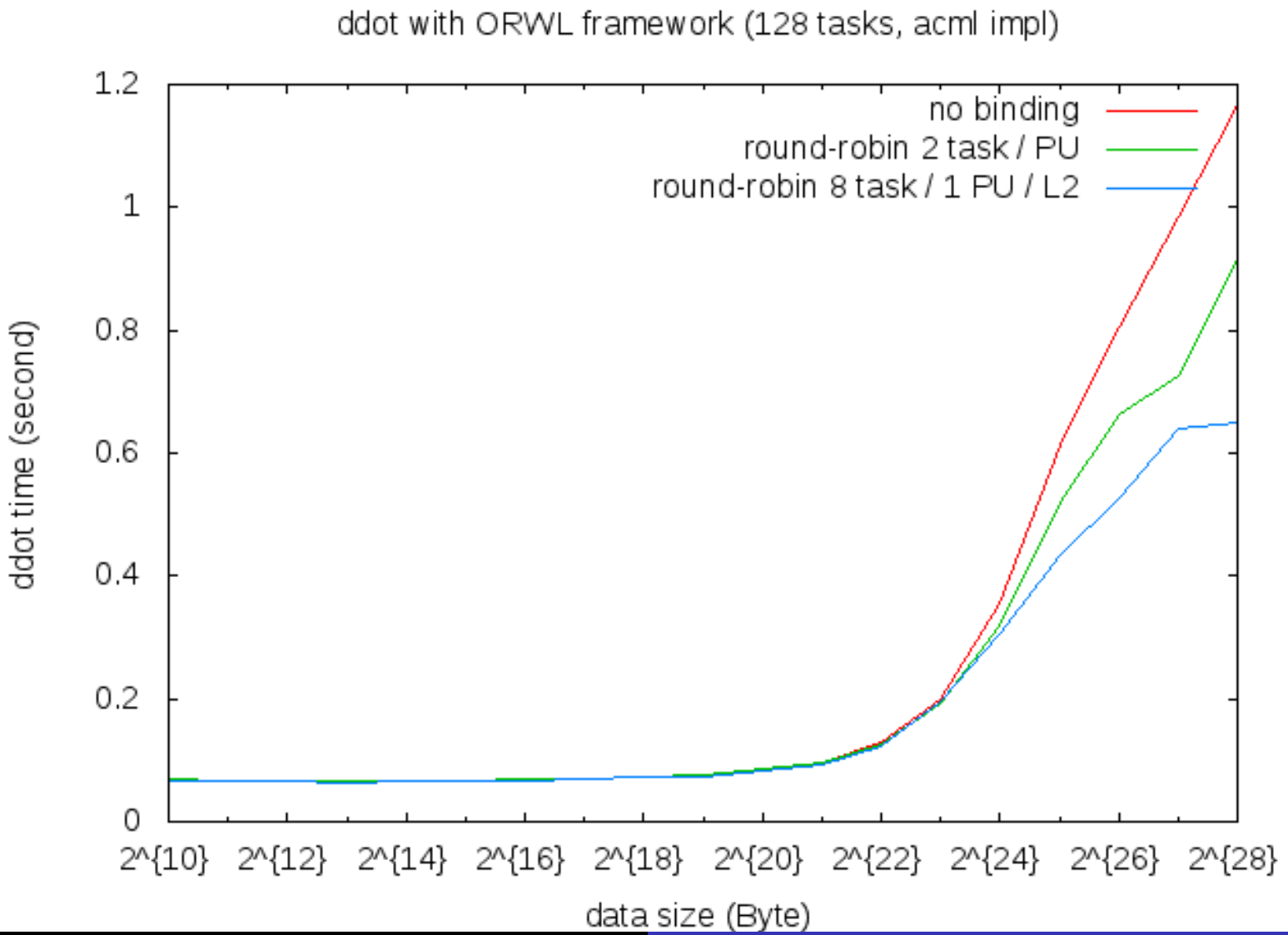

Machine (32GB)
Socket P#0 (32GB)
NUMANode P#0 (16GB)
L3 (6144KB)
L2 (2048KB)
L2 (2048KB)
L2 (2048KB)
L2 (2048KB)
L1 (16KB)
L1 (16KB)
L1 (16KB)
L1 (16KB)
L1 (16KB)
L1 (16KB)
L1 (16KB)
L1 (16KB)
Core P#0
PU P#0
Core P#1
PU P#4
Core P#2
PU P#8
Core P#3
PU P#12
Core P#4
PU P#16
Core P#5
PU P#20
Core P#6
PU P#24
Core P#7
PU P#28
NUMANode P#1 (16GB)
L3 (6144KB)
L2 (2048KB)
L2 (2048KB)
L2 (2048KB)
L2 (2048KB)
L1 (16KB)
L1 (16KB)
L1 (16KB)
L1 (16KB)
L1 (16KB)
L1 (16KB)
L1 (16KB)
L1 (16KB)
Core P#0
PU P#32
Core P#1
PU P#36
Core P#2
PU P#40
Core P#3
PU P#44
Core P#4
PU P#48
Core P#5
PU P#52
Core P#6
PU P#56
Core P#7
PU P#60

1 use case

- AMD Opteron(TM) Processor 6272 (64 Cores)
- initialization, ddot, reduce
- 128 tasks

Early Result

- Successful task binding
- Several tasks per CPU is better
- Binding 1 PU per L2 improves performance


ddot with ORWL framework (128 tasks, acml impl)
no binding
round-robin 2 task / PU
round-robin 8 task / 1 PU / L2
ddot time (second)
data size (Byte)
0
0.2
0.4
0.6
0.8
1
1.2
2^{10}
2^{12}
2^{14}
2^{16}
2^{18}
2^{20}
2^{22}
2^{24}
2^{26}
2^{28}

Conclusion

In this use case: simple resource restriction improves performance.

Future Work

- Multiply use cases
  - hardware
  - binding
  - application
- Identify bottlenecks in heterogeneous applications.
- Find an efficient algorithm (transparency).
- Eventually build a framework.

References I


Cédric Augonnet, Samuel Thibault, Raymond Namyst, and Pierre-André Wacrenier.

StarPU.


François Broquedis, Jérôme Clet-Ortega, Stéphanie Moreaud, Nathalie Furmento, Brice Goglin, Guillaume Mercier, Samuel Thibault, and Raymond Namyst.

hwloc: a Generic Framework for Managing Hardware Affinities in HPC Applications.


Pierre-Nicolas Clauss and Jens Gustedt.

Iterative Computations with Ordered Read-Write Locks.

References II

Thierry Gautier, Joao Vicente Ferreira Lima, Nicolas Maillard, and Bruno Raffin.

XKaapi: A Runtime System for Data-Flow Task Programming on Heterogeneous Architectures.

Emmanuel Jeannot, Guillaume Mercier, and François Tessier.

Process placement in multicore clusters: Algorithmic issues and practical techniques.

Laxmikant V. Kale and Sanjeev Krishnan.

Charm++: A portable concurrent object oriented system based on c++.

François Pellegrini and Jean Roman.

Scotch: A software package for static mapping by dual recursive bipartitioning of process and architecture graphs.