Parallel Programming at the Exascale Era: A Case Study on Parallelizing Matrix Assembly For Unstructured Meshes

Eric Petit, Loïc Thebault, Quang V. Dinh
May 2014
EXA2CT Consortium
Enhance Programming Models

WP2

Improve efficiency, robustness and scalability of GPI/GASPI, Shark, PATUS and leverage task parallelism

WP3

Proto-Applications

Extract representative proto-applications from HPC applications

WP1

Enhance Numerical Algorithms

Scalable and robust numerical solvers
The Proto-Application Concept

• Aka mini-app, proxy-app (NERSC trinity, Argonne CESAR...)

• **Our main tool in Exa2ct**

  • **Objectives:** Reproduce at scale the behavior of a set of HPC applications and support the development of optimizations that can be translated into the original applications
  • Easier to execute, modify and re-implement

  • If you cannot make the application open-source, you can at least open-source the problems.
    • Support community engagement
    • Reproducible and comparable results
    • Interface with application developers
Exascale Challenges for RT and Applications

- Increasing number of nodes, increasing number of cores, accelerators
  - Some resources do not scale
    - Memory per core, Bandwidth, Coherence protocol, Network interconnect, Fault tolerance
  - Frontier are becoming fuzzier => Heterogeneity
    - Distributed/shared? Software/hardware? "Core" definition? Compute capabilities, imbalance...
- Multiplication of hierarchical levels => Non uniformity
  - Different scales: BW, memory size, performance
  - Global events: barrier, broadcast, memory coherency
Exascale Challenges for RT and Applications

• **Evolutions are requested** for applications, runtimes and programming models
  • **Express the new resources usage**
    • Accelerators
    • Multi(many)-core
    • Large scale number of nodes
    • PROGRAMMABILITY and PORTABILITY (inc. Perf)
  • **Adapt to the hardware** design constraints
    • More concurrent resources
    • Limited BW and memory budget
    • Increasing need for fault-tolerance
    • Architecture oblivious/aware?
Exascale Challenges for RT and Applications

• **More concurrency**
  - Enough independant tasks
  - Communication overlap
  - Privatize memory to avoid communication (& sync)
  - Remember Amdhal: The more core, the higher is the proportion of the sequential code

• **More locality**
  - Memory
    • Core level, Socket level (including HWA), Network level
  - But also communications
    • Synchronisation, Data

• **We need both for performance scalability**
Objectives

- Express fine-grain dependencies of computations and communications using tasks to efficiently exploit the hundreds of cores of one Exascale node

- Benefit from automatic load balancing and tolerance to machine jitter of task parallel runtimes and asynchronous one sided

- Leverage divide-and-conquer algorithms to increase parallelism and data locality

- Improve the productivity of parallel programming
Divide and Conquer Parallelization of Finite Element Method Assembly

- **Objectives:**
  - Expose parallelism
    - Expose many independent task for many cores
  - Make an efficient use of shared memory
    - Save BW (Locality)
    - Save Memory (Avoid MPI buffer)
  - More efficient synchronisation
    - Prepare scaling on a larger number of cores
    - Keep synchronisation local
    - Use shared memory instead of MPI
  - Improve the load balancing
    - Fine grain parallelism
    - Work stealing/work sharing
Finite Element Method Assembly

- Basic FEM iteration:
  - Matrix A assembly from the mesh
  - Solve $Ax=b$
  - Update mesh (values, geometry)
- Building the linear system from the MESH
  - Elt-Node => Sparse matrix storage
  - Reduction of elements values on the edges and / or the nodes
Finite Element Method Assembly
2D Illustration

Reduction done on each edge from all neighbor elements

Edges update (+= reduction) must be sequential

\[ X_{ij} \neq 0 \text{ if there is an edge between } i \text{ and } j \]

(Very) Sparse and symmetric matrix
MPI Domain Decomposition

Distributed memory parallelism

Communicate interface elements

Efficient on current architectures

Sub-optimal on future architectures
  • Data duplications
  • Synchronizations
Elements sharing an edge have different colors.

Colors are computed sequentially.

Elements of a same color can be computed in parallel.

Simple to implement.

Bad temporal locality.
High memory bandwith requirements.
Global synchronizations.
The D&C approach

Left and right sub-domains are independant

Elements at the frontier form a separator
Separators must be treated after left and right sub-domains

Applied recursively to all sub-domains → Recursive Tree
The D&C approach

function compute (subdomain)  
    if Node is not a leaf 
        spawn compute (subdomain.left) 
        compute (subdomain.right) 
        sync 
        compute (subdomain.sep) 
    else 
        FEM_assembly (subdomain) 
    end
The D&C approach

- Optimizing locality: data permutation (composition of all recursive node permutation)
- Done only once (Topological partitioning)
The D&C approach

Can create many independant tasks

=> Concurrency

Leaves data set can be downsized at will to fit into caches

=> Data locality

Only one synchronization per task between the 2 local children

=> Sync locality

Only Log (N) synchronizations on the critical path

=> Sequential part minimization
The Dassault test case

DEFMESH Mesh Deformer from Dassault Aviation

- Fluid Dynamic Code based on Finite Element Method
- Non-Linear: displacements cut in small increments

3D Mesh: Airplane Fuel Tank

- Composed of 6,346,108 elements and 1,079,758 nodes
An Illustration of the D&C impact

• Initial
An Illustration of the D&C impact

• D&C 2
An Illustration of the D&C impact

- D&C 4
An Illustration of the D&C impact

• D&C 8
An Illustration of the D&C impact

- D&C 16
An Illustration of the D&C impact

- D&C 32
An Illustration of the D&C impact

- D&C 3000
- Leaves Size <50
Results: Execution Time

2 sockets of 6 cores Intel Xeon Westmere @ 2.67GHz
Results: Parallel Efficiency

2 sockets of 6 cores Intel Xeon Westmere @ 2.67GHz
Results: Performance

2 sockets of 6 cores Intel Xeon Westmere @ 2.67GHz

Last result, 3x improvement! (ivy-bridge)
Results: Locality

2 sockets of 6 cores Intel Xeon Westmere @ 2.67GHz
Impact on SPMV?

The original matrix was already optimized for locality however...
All the threads accessing the same lines (dup in cache)
Few threads accessing the whole vector

Bad load balancing
BW demanding:
- Bad locality
- Memory coherency
- Larger band
Impact on SPMV?

- Access a limited number of subdomains, one main very dense, others are very sparse separators.
- Increased locality.
- Minimized synchronisation.
- => Save BW.
Impact on SPMV?

- Improved locality on the matrix and the input vector. Conserve locality for the output vector.
- 10 to 20% improvement on the CG solver without editing a single line of code!
Conclusion 1/2

• The experiment on the DA test-case is successfull and ready to be used in DEFMESH production code.
• Since march the approach start be experimented in their main CFD code.
• Interesting integration process with all the parallelisation in C++ and the physics in FORTRAN
• A proto-app to demonstrate the D&C assembly to the community is almost ready to be open-sourced, stay tuned!
Conclusion 2/2

• Future works:
  • D&C friendly matrix format (CSB?) and SPMV, SPMM, SPTMM, SPMMM
  • Shared memory parallelized solvers
  • Impact of the partitionner (scotsh,metis,SAW)
  • Use D&C at the distributed level
    • Using GASPI/MPI3.0
  • Studying dynamic load balancing
    • Independant work
    • Tree based (D&C)
    • Symmetric execution (MIC/Host, starPU? Xkaapi?)
Thank you all for your attention!

• Question?
CONTACTS AND WORKPACKAGE LEADERS

COORDINATOR
Wilfried Verachtert, Imec
wilfried.verachtert@imec.be

PROGRAMMING MODELS
Eric Petit, UVSQ
eric.petit@prism.uvsq.fr

NUMERICAL ALGORITHMS
Wim Vanroose, U Antwerp
wim.vanroose@ua.ac.be

PROTO-APPLICATIONS
Marie-Christine Sawley, Intel
marie-christine.sawley@intel.com
Geometric vs. Topologic

Geometric Example

Topologic Example

Space

kd-tree

Mesh + Dual

Dual Graph Partitioning