WILL MIDDLEWARE AND DSL CHANGE THE PROGRAMMING HABITS WHEN COPING WITH EXASCALE MACHINES?
Motivations: an example
Manual optimization is becoming too difficult

HydroC miniapp: https://github.com/HydroBench/Hydro/tree/master/HydroC

Credit: Jason Sewall, Intel
HydroC optimizations done

- **Tiling**
  - Another name for blocking
  - Requires intra halo regions
  - Rolling buffers strategy to improve data locality

- **Vectorization**
  - Align data
  - Use intrinsics instead of auto vectorization

- **Arithmetic**
  - Optimize equations to reduce the number of / and sqrt()
Lessons learned here

• New architectures force code change on you
  • Or you’ll reach only a tiny fraction of % of the peak

• Data layout is important to exhibit more parallelism
  • Reduce the Amdahl’s impact

• Efficient Vectorization is difficult for a large code

• Questions:
  • What if I have an old code (legacy code)?
  • What can be done to simplify development of future codes?
Legacy codes
The heritage

• **Legacy codes survive for decades**
  - Some are still Fortran based (F90 usually)

• **Most are MPI only**
  - MPI + OpenMP is rare
  - Suboptimal behavior on a CPU with a large number of cores

• **Consequences**
  - CS issues have been mostly overlooked
    - NUMA effects
    - Thread safety
    - Reentrance
    - Cache usage
    - False sharing
    - Memory usage
    - …
A solution = circumvent some issues with a runtime

• **A clever runtime will**
  - Provide a seamless view of different models
    - Homogeneous thread based MPI + OpenMP
  - Hide some of the machine challenges
  - Allow for future extensions and experimentations
  - Provides a smooth transition method to exascale programming models

• **The MPC runtime is such a solution**
  - Developed at CEA and UVSQ
    - Active collaboration with TACC also
  - Web site maintained at University of Oregon
    - Coupled with TAU
  - Is used in some major CEA codes
    - Some positive feedbacks from DOE labs
MPC in a nutshell

• Multi-Processor Computing (MPC) framework
  Freely available at http://mpc.sourceforge.net (version 2.5.0)
  • Credit/ Contact: marc.perache@cea.fr, patrick.carribault@cea.fr or julien.jaeger@cea.fr

• Summary
  Unified parallel runtime for clusters of NUMA machines
  • Unification of several parallel programming models
    • MPI 1.3 compliant, thread based MPI, MPI_THREAD_MULTIPLE
    • POSIX Thread,
    • OpenMP 2.5 compliant
  • Integration with other HPC components
    • Parallel memory allocator for NUMA optimization
    • patched GCC
    • patched GDB, supported in Alinea DDT
    • HWLOC, …

• Extended TLS

• Hierarchical storage
What MPC can do for a legacy code

- NUMA effect

- MPI + OpenMP

- Global variable
  - F90 sources

- Memory footprint

- NUMA allocator

  - MPI_THREAD_MULTIPLE
  - memory optimization

  - TLS extensions

  - Automatic privatization

  - Hierarchical storage

  - One instance of physical databases on a node

A good runtime is a key piece of the future software stack
Preparing the future
How to avoid exposing the developers to gory details?

- **Two tracks are investigated at CEA**
  - Frameworks such as Arcane: In production and evolving
  - DSLs such as NABLA: An R&D effort

- **DSL ∇: Numerical Analysis-Based Language**
- Goal 1: how to make codes independent of machine generations
- Goal 2: how to simplify developments

- Separates the coding of the algorithm from the machine representation

- Uses the same source on different parallel architectures

- **Credit:** Jean-Sylvain.Camier@cea.fr, presented at PP14
The \textbf{∇} toolchain

Ligne de commande et de compilation

Mêmes sources \textbf{∇}

Analyse des sources

Arbre Syntaxique Abstrait (AST)

Optimisations communes de l'AST

Optimisation & Transformations

Caractérisation et Transformations

Génération vers ≠ cibles

Arcane: Seq, MPI, MPC, MPI+X

Materials

Prefetching

Caches Awareness

Mathematica: TCP+threads

HyODA

Cartesian

Particles

GMP

Aleph

Mail

Xeonφ: AVX, (TBB), (MPC), (MPI)

Prefetching

Caches Awareness

scatter/gather

Cartesian Xeonφ

AoS vs SoA

scatter/gather

nabla.h

nabla.cu

nabla.h

nabla.cc

nabla.h

nabla.axi

nabla.cxx

TrefleService.h

TrefleService.cc

GlaceModule.h

GlaceModule.cc

Config

main.cc

nablaconfig

nabla.cc

nabla.axi

nabla.m

[Image]
\( \nabla \) was able to express the following:

<table>
<thead>
<tr>
<th>Numerical schemes</th>
<th>Mini-Application</th>
<th>LOC ( \nabla )</th>
</tr>
</thead>
<tbody>
<tr>
<td>Explicit Lagrangian + indirections</td>
<td>LULESH\textsubscript{LLNL}</td>
<td>1030</td>
</tr>
<tr>
<td></td>
<td>GLACE\textsubscript{CEA}</td>
<td>266</td>
</tr>
<tr>
<td></td>
<td>MicroHydro\textsubscript{CEA}</td>
<td>217</td>
</tr>
<tr>
<td>Explicit Cartesian</td>
<td>Hydro (Sethi)\textsubscript{CEA}</td>
<td>757</td>
</tr>
<tr>
<td>Implicit</td>
<td>Schrödinger\textsubscript{CEA}</td>
<td>346</td>
</tr>
<tr>
<td></td>
<td>Diffusion DDFV\textsubscript{CEA}</td>
<td>264</td>
</tr>
<tr>
<td></td>
<td>Diffusion DIFF\textsubscript{CEA}</td>
<td>76</td>
</tr>
<tr>
<td>Monte-Carlo</td>
<td>MCTB\textsubscript{CEA}</td>
<td>828</td>
</tr>
<tr>
<td></td>
<td>MC1D\textsubscript{CEA}</td>
<td>276</td>
</tr>
<tr>
<td>Molecular Dynamics</td>
<td>CoMD\textsubscript{LANL}</td>
<td>293</td>
</tr>
<tr>
<td></td>
<td>MiniMD\textsubscript{SANDIA}</td>
<td>474</td>
</tr>
</tbody>
</table>
Performances validation of the $\nabla$ concept: LULESH

- Performances of the generated (C++, Cuda) ≅ «natives» optimized versions
- Strong-scaling of Lulesh 32768 cells  XeonX5560@2.80GHz (in s)

<table>
<thead>
<tr>
<th>32768 cells</th>
<th>Optimized</th>
<th>$\nabla$ Arcane + pthread</th>
<th>$\nabla$ Arcane + MPI</th>
<th>$\nabla$ Arcane + MPC</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>66</td>
<td>83</td>
<td>90</td>
<td>83</td>
</tr>
<tr>
<td>2</td>
<td></td>
<td>47</td>
<td>50</td>
<td>47</td>
</tr>
<tr>
<td>4</td>
<td></td>
<td>30</td>
<td>24</td>
<td>32</td>
</tr>
<tr>
<td>8</td>
<td></td>
<td>21</td>
<td>15</td>
<td>20</td>
</tr>
<tr>
<td>16</td>
<td></td>
<td>9</td>
<td>14</td>
<td></td>
</tr>
<tr>
<td>32</td>
<td></td>
<td>6</td>
<td>12</td>
<td></td>
</tr>
</tbody>
</table>

- Lulesh CUDA@Tesla = 9s vs $\nabla$-Lulesh = 11s

- $\nabla$ allows for an **incremental** approach of performance enhancement
Going a step further

Mixing a DSL and a runtime capabilities

What can be done if the DSL produces automatically arch independent kernels?
Goal: Harness at the same time CPUs and accelerators in the context of irregular numerical computations

- Balance workload between each architecture by introducing a two-level work stealing mechanism:

- Improve locality with a software cache strongly coupled to the scheduler
  - Designed to reduce memory transfers by retaining data in off-chip memory
  - Scheduler guided by cache affinity to avoid unnecessary transfers
Heterogeneous Scheduler

LU Decomposition with Dense & Sparse Blocks, cumulated perf. of step 3 (SGEMM → MKL & CUBLAS) vs Matrix size

2x AMD 6164HE (24 cores @ 1.7 GHz)
1x Nvidia Geforce GTX 470 (448 cores @ 1.215GHz)

> Paper presented at MULTIPROG (January 2012)

Jean-Yves Vet, Patrick Carribault, Albert Cohen, Multigrain Affinity for Heterogeneous Work Stealing, MULTIPROG ‘12

> Could be used to exploit several types of many-core processors (Nvidia GPUs, AMD GPUs/APUs, Intel MIC, …)
Conclusions

• Getting at the exascale level will require major code evolutions
  Is it (it is) time to consider other tracks? (!)

• Proper runtimes will help for the legacy codes
  MPC is a good starting point

• DSLs are a really promising way
  Our ∇ experiment confirms other results (Liszt)
  Not ready for massive production yet

• In the meantime frameworks are the safer option
  In conjunction with their tools for increased software productivity

• Next step = a DSL working in collaboration with the runtime
The future is a multi-disciplinary team + a framework and tools for increased productivity.
Bibliography

• M. Pérache, P. Carribault, H. Jourdren, *MPC-MPI: An MPI Implementation Reducing the Overall Memory Consumption* (EuroPVM/MPI'09)
• P. Carribault, M. Pérache, H. Jourdren, *Enabling Low-Overhead Hybrid MPI/OpenMP Parallelism with MPC* (IWOMP'10)
• P. Carribault, M. Pérache, H. Jourdren, *Thread-Local Storage Extension to Support Thread-Based MPI/OpenMP Applications* (IWOMP’11)
• M. Tchiboukdjian, P. Carribault, M. Pérache, *Hierarchical Local Storage: Exploiting Flexible User-Data Sharing Between MPI Tasks*, (IPDPS’12)
• Contemporary High Performance Computing, from Petascale toward Exascale, chap 4 Tera 100, Ed. Jeffrey S. Vetter