BOAST

A Generative Language for Intensive Computing Kernels

Porting HPC applications to the Mont-Blanc Prototype

Kevin Pouget, Brice Videau, Jean-François Méhaut
UJF-CEA, CNRS, INRIA, LIG

International Laboratory LICIA: UFRGS and LIG
Associated INRIA team ExaSE: UFRGS, USP, PUC Minas

EU FP7 IRSES HPC GA: INRIA, BCAM, UFRGS, UNAM, BRGM, ISTerre
EU FP7 ICT Mont-Blanc: BSC, BULL, ARM, CNRS, INRIA, CINECA, Bristol, JSC, HLRS, LRZ...

Maison de la simulation, Saclay
March 9, 2015
Introduction
Joint Project-Team (Inria, Grenoble INP, UJF) in the LIG laboratory @ GIANT/Minatec

- Fabrice Rastello, Florent Bouchez Tichadou, François Broquedis, Frédéric Desprez, Yliès Falcone, Jean-François Méhaut
- 8 PhD, 3 Post-doc, 1 Engineer
Florent Bouchez Tichadou  MdC UJF (PhD Lyon 2009, 1Y Bangalore, 3Y Kalray, NANOsim) compiler optimization, compiler back-end

François Broquedis  MdC INP (PhD Bordeaux 2010, 1Y MESCAL, 3Y MOAIS) runtime systems, OpenMP, memory management

Frédéric Desprez  (DR1 INRIA: Graal, Avalon) parallel algorithmic, numerical libraries

Ylies Falcone  MdC UJF (PhD Grenoble 2009, 2Y Rennes, Vasco) validation, enforcement, debugging, runtime

Jean-François Mehaut  Pr UJF (MESCAL, NANOsim) runtime, debugging, memory management, scientific applications

Fabrice Rastello  CR1 INRIA (PhD Lyon 2000, 2Y STMicro, COMPSYS, GCG) compiler optimization, graph theory, compiler back-end, automatic parallelization
Overall Objectives

Domain: Compiler optimization and runtime systems for performance and energy consumption (not liability, nor WCET)

Issues: Scalability and heterogeneity/complexity = trade-off between specific optimizations and programmability/portability

Target architectures: VLIW / SIMD / embedded / many-cores / heterogeneity

Applications: dynamic-systems / loop-nests / graph-algorithmic / signal-processing

Approach: combine static/dynamic & compiler/run-time
Low-Power High Performance Computing

• **Industrial collaboration**

• **Kalray** ([http://www.kalray.eu](http://www.kalray.eu))
  • French fabless semiconductor and software company founded in 2008
  • France (Grenoble, Orsay), USA (California), Japan (Tokyo)
  • Company developing and selling a new generation of manycore processors

• **MPPA-256**
  • Multi-Purpose Processor Array (MPPA)
  • Manycore processor: 256 cores on a single chip
  • Low Power Consumption [5W - 11W]
Kalray MPPA-256 architecture

- 256 cores (PEs) @ 400 MHz: 16 clusters, 16 PEs per cluster
- PEs share 2MB of memory
- Absence of cache coherence protocol inside the cluster
- Network-on-Chip (NoC): communication between clusters
- 4 I/O subsystems: 2 connected to external memory
Seismic Wave Propagation (Ondes3D, BRGM)

- Simulation composed by **time steps**
- **In each time step (3D simulation)**
  - The first triple nested loop computes the **velocity** components
  - The second loop reuses velocity result of the previous time step to update the **stress** field

```plaintext
for x ← 1 to x_dimension
    for y ← 1 to y_dimension
        do { for z ← 1 to z_dimension
               do { COMPUTE VELOCITY() }
        }
for x ← 1 to x_dimension
    for y ← 1 to y_dimension
        do { for z ← 1 to z_dimension
               do { COMPUTE STRESS() }
        }

4th-order stencil
```
Overview of Parallel Execution on MPPA-256

- **Two-level tiling scheme** to exploit the memory hierarchy of MPPA-256
Results - Input problem size of 2 GB

![Bar chart showing energy and time-to-solution for different hardware configurations.]

- Xeon E5 (8 cores)
- Quadro K4000 (768 cores)
- MPPA-256 (256 cores)

Energy-to-solution (J):
- Xeon E5: 3892 J
- Quadro K4000: 2609 J
- MPPA-256: 100.2 J

Time-to-solution (s):
- Xeon E5: 61.0 s
- Quadro K4000: 46.2 s
- MPPA-256: 152 s

3.5x improvement in time-to-solution with MPPA-256 compared to Xeon E5.
Results - Input problem size of 2 GB

<table>
<thead>
<tr>
<th></th>
<th>Energy-to-solution (J)</th>
<th>Time-to-solution (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Xeon E5 (8 cores)</td>
<td>3892</td>
<td>61.0</td>
</tr>
<tr>
<td>Quadro K4000 (768 cores)</td>
<td>2609</td>
<td>46.2</td>
</tr>
<tr>
<td>MPPA-256 (256 cores)</td>
<td>100.2</td>
<td>75.2</td>
</tr>
</tbody>
</table>

The graph shows a comparison of energy and time-to-solution for three different hardware configurations, with a 5.2x improvement in time-to-solution.
Results - Input problem size of 2 GB

![Bar chart showing energy-to-solution for different systems.]

- Xeon E5 (8 cores): Energy 3892 J, Time 61.0 s
- Quadro K4000 (768 cores): Energy 2609 J, Time 46.2 s
- MPPA-256 (256 cores): Energy 100.2 J, Time 152 s

The time-to-solution is 2.2x slower for MPPA-256 compared to Quadro K4000.
On the energy efficiency and performance of irregular application executions on multicore, NUMA and manycore platforms

Emilio Francesquini, Márcio Castro, Pedro H. Penna, Fabrice Dupros, Henrique C. Freitas, Philippe O.A. Navaux, Jean-François Méhaut

HIGHLIGHTS
- Programming for a manycore is challenging.
- Limited memory and NoC are among the most important constraints of manycores.
- For CPU-bound and mixed workloads, MPPA-256 achieves better performance than Xeon.
- MPPA-256 consumes up to 13× less energy than embedded and general-purpose multicore systems.

ARTICLE INFO
Article history:
Received 24 March 2014
Received in revised form 17 October 2014
Accepted 4 November 2014
Available online 13 November 2014

ABSTRACT
Until the last decade, performance of HPC architectures has been almost exclusively quantified by their processing power. However, energy efficiency is being recently considered as important as raw performance and has become a critical aspect to the development of scalable systems. These strict energy constraints guided the development of a new class of so-called light-weight manycore processors. This study evaluates the computing and energy performance of two well-known irregular NP-hard problems – the Traveling-Salesman Problem (TSP) and K-Means clustering – and a numerical seismic simulation, comparing 128 core MPPA-256, Xeon E5-2670 and an embedded processor.
In the past, waiting for a new generation of hardware was enough to obtain performance gains.

Nowadays, architecture are so different that performance regress when a new architecture is released.

Sometimes the code is not fit anymore and cannot be compiled.

Few applications can harness the power of current computing platforms.

Thus, optimizing scientific application is of paramount importance.
Multiplicity of Computing Architectures

A High performance Computing application can encounter several types of architectures:

- General-Purpose Multicore CPU (Intel, AMD, PowerPC...)
- Graphical Accelerators (NVIDIA, ATI-AMD, ...)
- Computing Accelerators (Xeon Phi, MPPA-256, Tilera, CELL...)
- Low power CPUs (ARM...)

Those architectures can present drastically different characteristics.
## Architectures Comparison

<table>
<thead>
<tr>
<th>Architecture</th>
<th>AMD/Intel CPUs</th>
<th>ARM</th>
<th>GPUs</th>
<th>Xeon Phi</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cores</td>
<td>4-12</td>
<td>2-4</td>
<td>512-2496</td>
<td>60</td>
</tr>
<tr>
<td>Cache</td>
<td>3l</td>
<td>2l</td>
<td>2l incoherent</td>
<td>2l</td>
</tr>
<tr>
<td>Memory (GiB)</td>
<td>2-4 (per core)</td>
<td>1 (per core)</td>
<td>2-6</td>
<td>6-16</td>
</tr>
<tr>
<td>Vector Size</td>
<td>2-4</td>
<td>1-2</td>
<td>1-2</td>
<td>8</td>
</tr>
<tr>
<td>Peak GFLOPS</td>
<td>10-20 (per core)</td>
<td>2-4 (per core)</td>
<td>500-1500</td>
<td>1000</td>
</tr>
<tr>
<td>Peak GiB/s</td>
<td>20-40</td>
<td>2.5-5</td>
<td>150-250</td>
<td>200</td>
</tr>
<tr>
<td>TDP W</td>
<td>75</td>
<td>5</td>
<td>200</td>
<td>300</td>
</tr>
<tr>
<td>GFLOPS/W</td>
<td>1-3</td>
<td>2-4</td>
<td>2-7</td>
<td>3</td>
</tr>
</tbody>
</table>

**Table:** Comparison between commonly found architectures in HPC.
Exploiting Various Architectures

Usually work is done on a class of architecture (CPUs or GPUs or accelerators).

Well Known Examples

- Atlas (Linear Algebra CPU)
- PhiPAC (Linear Algebra CPU)
- Spiral (FFT CPU)
- FFTW (FFT CPU)
- NukadaFFT (FFT GPU)

No work targeting several class of architectures. What if the application is not based on a library?
The Mont-Blanc European Projects


- Develop prototypes of HPC clusters using low power commercially available embedded technology (ARM CPUs, low power GPUs...).
- Design the next generation in HPC systems based on embedded technologies and experiments on the prototypes.
- Develop a portfolio of existing applications to test these systems and optimize their efficiency, using BSC’s OmpSs programming model (11 existing applications were selected for this portfolio).
- Build Software Stack (OS, runtime, performance tools,...)

Prototype: based on Exynos 5250: ARM dual core Cortex A15 with T604 Mali GPU (OpenCL)
**BigDFT a Tool for Nanotechnologies**

Ab initio simulation:

- Simulates the properties of crystals and molecules,
- Computes the electronic density, based on Daubechies wavelet.

This formalism was chosen because it is fit for HPC computations:

- Each orbital can be treated independently most of the time,
- Operator on orbitals are simple and straightforward.

Mainly developed in Europe:

- CEA-DSM/INAC (Grenoble)
- Basel, Louvain la Neuve,...

Electronic density around a methane molecule.
BigDFT as an HPC application

Implementation details:
- 200,000 lines of Fortran 90 and C
- Supports MPI, OpenMP, CUDA and OpenCL
- Uses BLAS
- Scalability up to 16000 cores of Curie and 288GPUs

Operators can be expressed as 3D convolutions:
- Wavelet Transform
- Potential Energy
- Kinetic Energy

These convolutions are separable and filter are short (16 elements). Can take up to 90% of the computation time on some systems.
SPECFEM3D a tool for wave propagation research

Wave propagation simulation:

- Used for geophysics and material research,
- Accurately simulate earthquakes,
- Based on spectral finite element.

Developed all around the world:

- France (CNRS Marseille),
- Switzerland (ETH Zurich) CUDA,
- United States (Princeton) Networking,
- Grenoble (LIG/CNRS) OpenCL.
SPECFEM3D as an HPC application

Implementation details:

- 80,000 lines of Fortran 90
- Supports MPI, CUDA, OpenCL and an OMPSs + MPI miniapp
- Scalability up to 693,600 cores on IBM BlueWaters
Talk Outline

2 Case Studies

3 A Parametrized Generator

4 Evaluation

5 Conclusions and Future Work
Case Study 1: BigDFT’s MagicFilter

The simplest convolution found in BigDFT, corresponds to the potential operator.

Characteristics
- Separable,
- Filter length 16,
- Transposition,
- Periodic,
- Only 32 operations per element.

Pseudo code

```c
double filt[16] = {F0, F1, ..., F15};
void magicfilter(int n, int ndat,
                 double *in, double *out){
    double temp;
    for(j=0; j<ndat; j++) {
        for(i=0; i<n; i++) {
            temp = 0;
            for(k=0; k<16; k++) {
                temp += in[ ((i-7+k)%n) + j*n ] * filt[k];
            }
            out[j + i*ndat] = temp;
        } }
```

BOAST
Case study 2: SPECFEM3D port to OpenCL

Existing CUDA code:

- 42 kernels and 15000 lines of code
- Kernels with 80+ parameters
- \(\sim\) 7500 lines of cuda code
- \(\sim\) 7500 lines of wrapper code

Objectives:

- Factorize the existing code,
- Single OpenCL and CUDA description for the kernels,
- Validate without unit tests, comparing native Cuda to generated Cuda executions
- Keep similar performances.
A Parametrized Generator
### Classical Software Development Loop

- **Kernel optimization workflow**
- **Usually performed by a knowledgeable developer**
Classical Software Development Loop

- Compilers perform optimizations
- Architecture specific or generic optimizations
Classical Software Development Loop

- Performance data hint at source transformations
- Architecture specific or generic hints
Multiplication of kernel versions or loss of versions

Difficulty to benchmark versions against each-other
BOAST Development Loop

- Meta-programming of optimizations in BOAST
- High level object oriented language
BOAST Development Loop

- Generate combination of optimizations
- C, OpenCL, FORTRAN and CUDA are supported
Compilation and analysis are automated

Selection of best version can also be automated
BOAST

Application kernel (SPECFEM3D, BigDFT, ...)

Optimization space prunner: ASK, Collective Mind

Binary analysis tool like MAQAO

Kernel written in BOAST DSL

Performance measurements

Binary kernel

Select target language

Select input data

Select compiler and options

Select performance metrics

Best performing version

BOAST code generation

BOAST runtime

gcc, opencl

C kernel

Fortran kernel

OpenCL kernel

CUDA kernel

C with vector intrinsics kernel
Use Case Driven

Parameters arising in a convolution:

- Filter: length, values, center.
- Direction: forward or inverse convolution.
- Boundary conditions: free or periodic.
- Unroll factor: arbitrary.

How are those parameters constraining our tool?
Features required

Unroll factor :
  - Create and manipulate an unknown number of variables,
  - Create loops with variable steps.

Boundary conditions :
  - Manage arrays with parametrized size.

Filter and convolution direction :
  - Transform arrays.

And of course be able to describe convolutions and output them in different languages.
Proposed Generator

Idea: use a high level language with support for operator overloading to describe the structure of the code, rather than trying to transform a decorated tree.

Define several abstractions:

- Variables: type (array, float, integer), size...
- Operators: affect, multiply...
- Procedure and functions: parameters, variables...
- Constructs: for, while...
### Sample Code: Variables and Parameters

```plaintext
# simple Variable
i = Int "i"

# simple constant
lowfil = Int( "lowfil", :const => 1-center )

# simple constant array
fil = Real("fil", :const => arr, :dim => [ Dim(lowfil, upfil) ])

# simple parameter
ndat = Int("ndat", :dir => :in)

# multidimensional array, an output parameter
y = Real("y", :dir => :out, :dim => [ Dim(ndat), Dim(dim_out_min, dim_out_max) ])
```

Variables and Parameters are objects with a name, a type, and a set of named properties.
Sample Code: Procedure Declaration

The following declaration:

```plaintext
1 p = Procedure("magic_filter", [n, ndat, x, y], [lowfil, upfil])
2 open p
```

Outputs Fortran:

```plaintext
1 subroutine magicfilter(n, ndat, x, y)
2   integer(kind=4), parameter :: lowfil = -8
3   integer(kind=4), parameter :: upfil = 7
4   integer(kind=4), intent(in) :: n
5   integer(kind=4), intent(in) :: ndat
6   real(kind=8), intent(in), dimension(0:n-1, ndat) :: x
7   real(kind=8), intent(out), dimension(ndat, 0:n-1) :: y
```

Or C:

```plaintext
1 void magicfilter(const int32_t n, const int32_t ndat, const double * x, double * y){
2   const int32_t lowfil = -8;
3   const int32_t upfil = 7;
```
Sample Code: Constructs and Arrays

The following declaration:

```c
unroll = 5
pr For(j,1,ndat-(unroll-1), unroll) {
    #.....
    pr tt2 === tt2 + x[k,j+1]*fil[1]
    #.....
}
```

Outputs Fortran:

```fortran
do j=1, ndat-4, 5
    !......
    tt2=tt2+x(k,j+1)*fil(1)
    !......
endo
```

Or C:

```c
for(j=1; j<=ndat-4; j+=5){
    /*.............*/
    tt2=tt2+x[k-0+(j+1-1)*(n-1-0+1)]*fil[1-lowfil];
    /*.............*/
}
```
Generator Evaluation

Back to the test cases:

- The generator was used to unroll the Magicfilter and evaluate its performance on an ARM processor and an Intel processor.
- The generator was used to describe SPECFEM3D kernel.
Performance Results

**Tegra2**

**Cache access**

![Cache access graph for Tegra2](image1)

**Total cycles**

![Total cycles graph for Tegra2](image2)

**Intel T7500**

**Cache access**

![Cache access graph for Intel T7500](image3)

**Total cycles**

![Total cycles graph for Intel T7500](image4)
BigDFT Synthesis Kernel

Synthesys Speedup

function of unrolling factor
Improvement for BigDFT

- Most of the convolutions have been ported to BOAST.
- Results are encouraging: on the hardware BigDFT was hand optimized for, convolutions gained on average between 30 and 40% of performance.
- MagicFilter OpenCL versions tailored for problem size by BOAST gain 10 to 20% of performance.
SPECFEM3D OpenCL port

Fully ported to OpenCL with comparable performances (using the `global_s362ani_small` test case):

- On a 2*6 cores (E5-2630) machine with 2 K40, using 12 MPI processes:
  - OpenCL: 4m15s
  - CUDA: 3m10s

- On an 2*4 cores (E5620) with a K20 using 6 MPI processes:
  - OpenCL: 12m47s
  - CUDA: 11m23s

Difference comes from the capacity of CUDA to specify the minimum number of blocks to launch on a multiprocessor. Less than 4000 lines of BOAST code (7500 lines of CUDA originally).
Conclusions

Generator has been used to test several loop unrolling strategies in BigDFT.

Highlights:

- Several output languages.
- All constraints have been met.
- Automatic benchmarking framework allows us to test several optimization levels and compilers.
- Automatic non regression testing.
- Several algorithmically different versions can be generated (changing the filter, boundary conditions...).
Future Works and Considerations

Future work:

- Produce an autotuning convolution library.
- Implement a parametric space explorer or use an existing one (ASK: Adaptative Sampling Kit, Collective Mind...).
- Vector code is supported, but needs improvements.
- Test the OpenCL version of SPECFEM3D on the Mont-Blanc prototype.

Question raised:

- Is this approach extensible enough?
- Can we improve the language used further?