BOAST
Performance Portability Using Meta-Programming and Auto-Tuning

Brice Videau\(^2\), Kevin Pouget\(^1\), Luigi Genovese\(^2\), Thierry Deutsch\(^2\), Julien Bigot\(^5\), Guillaume Latu\(^4\), Virginie Grandgirard\(^4\), Dimitri Komatitsch\(^3\), Frédéric Desprez\(^1\), Jean-François Méhaut\(^1\)

\(^1\)INRIA/LIG - CORSE, \(^2\)CEA - L_Sim, \(^3\)CNRS, \(^4\)CEA - IRFM, \(^5\)CEA - Maison de la Simulation

Journées SUCCES, Grenoble
October 17, 2017
Scientific Application Portability

Limited Portability

- Huge codes (more than 100,000 lines), Written in FORTRAN or C++
- Collaborative efforts
- Use many different programming paradigms (OpenMP, OpenCL, CUDA, ...)

But Based on Computing Kernels

- Well defined parts of a program
- Compute intensive
- Prime target for optimization

Kernels Should Be Written

- In a portable manner
- In a way that raises developer productivity
- To present good performance
**HPC Architecture Evolution**

**Very Rapid and Diverse, Top500:**

- Sunway processor (TaihuLight)
- Intel processor + Xeon Phi (Tianhe-2)
- AMD processor + nVidia GPU (Titan)
- IBM BlueGene/Q (Sequoia)
- Fujitsu SPARC64 (K Computer)
- Intel processor + nVidia GPU (Tianhe-1)
- AMD processor (Jaguar)

**Tomorrow?**

- ARM + DSP?
- Intel Atom + FPGA?
- Quantum computing?

How to write kernels that could adapt to those architectures? (well maybe not quantum computing...)
Related Work

- **Ad hoc autotuners (usually for libraries):**
  - Atlas [6] (C macro processing)
  - SPIRAL [4] (DSL)
  - ...

- **Generic frameworks using annotation systems:**
  - POET [7] (external annotation file)
  - Orio [3] (source annotation)
  - BEAST [1] (Python preprocessor based, embedded DSL for optimization space definition/pruning)

- **Generic frameworks using embedded DSL:**
  - Halide [5] (C++, not very generic, 2D stencil targeted)
  - Heterogeneous Programming Library [2] (C++)
Classical Tuning of Computing Kernels

- Kernel optimization workflow
- Usually performed by a knowledgeable developer
Compilers perform optimizations

Architecture specific or generic optimizations
Classical Tuning of Computing Kernels

- Performance data hint at source transformations
- Architecture specific or generic hints
Classical Tuning of Computing Kernels

- Multiplication of kernel versions and/or loss of versions
- Difficulty to benchmark versions against each-other
Meta-programming of optimizations in BOAST

High level object oriented language
**BOAST Workflow**

- 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 Architecture

1. Select target language
2. Select optimizations
3. Select compiler and options
4. Select performance metrics
5. BOAST code generation

- Application kernel (SPECFEM3D, BigDFT, ...)
- Kernel written in BOAST DSL
- Optimization space pruner: ASK, Collective Mind
- Binary analysis tool like MAQAO
- Binary kernel
- Performance measurements
- Select input data
- Select target language
- Select optimizations
- Select compiler and options
- Select performance metrics
- BOAST runtime
- gcc, opencl

Best performing version:
- C kernel
- Fortran kernel
- OpenCL kernel
- CUDA kernel
- C with vector intrinsics kernel
**Gysela 2d Advection**

**Gysela: Gyrokinetic Semi-Lagrangian**

Tokamak plasma simulation for fusion (ITER)

- **Preparation steps**
  - Extract 4 targeted routines from Gysela (subpart of 2d advection)
  - Change **API** of the 2d advection kernel
    - only arrays of integers and floats for inputs/outputs
      - (transmitting data structures is possible but more complex)
  - Define valid *fake* inputs for the kernel to design a regression test
  - Integrate the reference/original version into BOAST

- **Install ruby & BOAST on 4 parallel machines**
  - Easiest step
  - Get a working compilation/execution of the kernel: a bit more difficult

- **Write a meta-program that *prints* a program**
  1. Need to learn a little bit of ruby & BOAST
  2. **Incremental approach:** begin with internal routines then external
  3. Identify what are the *parameters* of the auto-tuning
  4. Integrate the best kernel version to the Gysela compilation process
Gysela 2d advection (2)

- Auto-tuning parameters that we chose
  - directive based inlining / BOAST driven inlining
  - BOAST driven loop unrolling
  - C or Fortran code generated
  - scan versions of gfortran/gcc/icc/ifort (module load)
  - loop blocking parameter (one of the most internal loop)
  - explicit vectorization: BOAST generates INTEL intrinsics, e.g.
    ```
    ftmp1 = _mm256_setzero_pd( );
    ftmp2 = _mm256_setzero_pd( );
    ftmp1 = _mm256_fmadd_pd( base1[0], _mm256_load_pd( &ftransp[(0) * (4)] );
    ftmp2 = _mm256_fmadd_pd( base1[0 + 1], _mm256_load_pd( &ftransp[(0 + 1) * (4)] ), ftmp2 );
    ```

- Final result
  - ruby code of 200 lines for the 2d advection kernel
    compared to original fortran code of 300 lines

- Auto-tuning runs
  - configure the list of modules/compilers for the parameter scan
  - between 1 min and 20 min for the parameter scan on 1 machine
Auto-tuning on INTEL Westmere (2011)

Auto-tuning for 2D advection
Computing center at Marseille
12-cores node -
Intel X5675, 3.07GHz

Nb of runs in this scan: 609
Runs sorted from quickest to slowest
Result of the scan
(best parameters):

:lang: FORTRAN
:unroll: true
:force.inline: true
:intrinsic: false
:blocking.size: 4
:module: intel/16.0.2

Speedup: 1.9
Auto-tuning on INTEL Sandy-Bridge (2012)

Auto-tuning for 2D advection
Computing center at Orsay
16-cores node -
Intel E5-2670 v1, 2.60GHz

Result of the scan
(best parameters):

:lang: FORTRAN
:unroll: false
:force.inline: false
:intrinsic: false
:blocking.size: 2
:module: intel/15.0.0

Speedup: 1.7

2D advection kernel, averaged execution time

BOAST parameter scan / sorted
reference execution time
Auto-tuning on INTEL Haswell (2015)

Auto-tuning for 2D advection
Computing center at Montpellier
24-cores node-
   Intel E5-2690 v3, 2.60GHz

Result of the scan
(best parameters):

:lang: FORTRAN
:unroll: true
:force.inline: true
:intrinsic: false
:blocking.size: 4
:module: intel/14.0.4.211

Speedup: 2.0
Auto-tuning for 2D advection
Computing center at Montpellier
64-cores node -
   Intel 7210 1.30GHz

Result of the scan
(best parameters):
:lang: FORTRAN
:unroll: true
:force.inline: true
:intrinsic: false
:blocking.size: 32
:module: intel/17.0

Speedup: 3.6
Novel approach for DFT computation based on Daubechies wavelets

Fortran and C code, MPI, OpenMP, supports CUDA and OpenCL

Reference is hand tuned code on target architecture (Nehalem)

Toward a BLAS-like library for wavelets
SPECFEM3D

- Seismic wave propagation simulator
- SPECFEM3D ported to OpenCL using BOAST
  - Unified code base (CUDA/OpenCL)
  - Refactoring: kernel code base reduced by 40%
  - Similar performance on NVIDIA Hardware
  - Non regression test for GPU kernels
- On the Mont-Blanc prototype:
  - OpenCL+MPI runs
  - Speedup of 3 for the GPU version
Conclusions

- BOAST v2.0 is released
- BOAST language features:
  - Unified C and FORTRAN with OpenMP support,
  - Unified OpenCL and CUDA support,
  - Support for vector programming.
- BOAST runtime features:
  - Generation of parametric kernels,
  - Parametric compilation,
  - Non-regression testing of kernels,
  - Benchmarking capabilities (PAPI support)
  - Co-execution and numa-aware capabilities (using hwloc)
Perspectives

- Ongoing work on other applications: Alya, dgtdNano3d
- Couple BOAST with other tools:
  - Parametric space pruners (speed up optimization),
  - Binary analysis (guide optimization, MAQAO),
  - Source to source transformation (improve optimization),
  - Binary transformation (improve optimization).
- Improve BOAST:
  - Improve the eDSL to make it more intuitive,
  - Better vector support,
  - Gather feedback.
And the Future?

New architectures:

- **FPGAs:**
  - Supported via OpenCL,
  - longer compile time,
  - parallel compilation?

- **New vector architectures:**
  - Intel KNL and onward: masked vector instructions,
  - ARM SVE: meta programming is in the instruction set.

- **New memory architectures:**
  - 3D stacked high performance memory (KNL, GPUs): new address space,
  - Non Volatile RAM: new address space again (relevant for computing kernels?)?
Bibliography

Hartwig Anzt, Blake Haugen, Jakub Kurzak, Piotr Luszczek, and Jack Dongarra.
Experiences in autotuning matrix multiplication for energy minimization on gpus.
cpe.3516.

Jorge F. Fabeiro, Diego Andrade, and Basilio B. Fraguela.
Writing a performance-portable matrix multiplication.

Albert Hartono, Boyana Norris, and Ponnuswamy Sadayappan.
Annotation-based empirical performance tuning using Orio.
Also available as Preprint ANL/MCS-P1556-1008.

Markus Püschel, José MF Moura, Bryan Singer, Jianxin Xiong, Jeremy Johnson, David Padua, Manuela Veloso, and Robert W Johnson.
SPIRAL: A generator for platform-adapted libraries of signal processing algorithms.

Halide: a language and compiler for optimizing parallelism, locality, and recomputation in image processing pipelines.

R. Clint Whaley and Antoine Petitet.
Minimizing development and maintenance costs in supporting persistently optimized BLAS.

Qing Yi, Keith Seymour, Haihang You, Richard Vuduc, and Dan Quinlan.
POET: Parameterized optimizations for empirical tuning.