Automated Code Generation for Lattice QCD Simulation

Denis Barthou

INRIA Runtime

- Team within INRIA and CNRS/University of Bordeaux.
- 11 researchers, ass. prof, prof.
- *Runtime systems perform dynamically what cannot be done statically*
  - Interface between compilers/runtimes
  - Runtime support for heterogeneous architectures

Automatic code generation for LQCD

ANR PetaQCD project (2009-2013): a pluridisciplinary project around LQCD

- Physics (LAL and LPT, CNRS IN2P3, Orsay)
- Compiler/Runtime (CAPS, U. Bordeaux, INRIA Bordeaux and Saclay)
- Computer architecture (INRIA Rennes)

Among the objectives:

- Design a high level language for LQCD for writing new algorithms
  $\rightarrow$ a DSL, QIRAL
- Focus on the inversion of Dirac matrix

Still active community and development.

D. Barthou et al., Automated Code Generation for Lattice QCD and Beyond, CoRR, abs/1401.2039, 2014

Problem Formulation: Inverting Dirac

Mathematical formulation

Given η and *Dirac*, the Dirac matrix, compute vector ϕ:

$$Dirac\phi = \eta$$

Vectors of spinors, over 4D lattice. Dirac is sparse.

Usual approach: preconditionings + iterative Krylov methods.

Problem Formulation: Inverting Dirac

Mathematical formulation

Given η and *Dirac*, the Dirac matrix, compute vector ϕ:

$$Dirac\phi = \eta$$

Vectors of spinors, over 4D lattice. Dirac is sparse.
Usual approach: preconditionings + iterative Krylov methods.

Choice of the method

- Large zoo of iterative methods (CG, GCR, deflation, ...)
- Many possible preconditionings
  - Active field in research, new methods appear regularly
  - Many parameters to explore, possible combinations of algos.

Problem Formulation: Inverting Dirac

Where is parallelism ?

- Not in the methods
- In the structure of the sparse matrix

Problem Formulation: Inverting Dirac

Where is parallelism ?

- Not in the methods
- In the structure of the sparse matrix

Code generation techniques

- Using domain-specific parallel libraries (Quda, QDP++, Chroma, ...)
- Hand-tuned codes (MILC, ETMC, ...)
  - Many hand-tuned optimizations of formulas, using knowledge of *Dirac* and of preconditioning.
  - Many hand-tuned code optimizations achieved according to an **implicit** compiler/architecture and performance model.

Problem Formulation: Inverting Dirac

Where is parallelism ?

- Not in the methods
- In the structure of the sparse matrix

Code generation techniques

- Using domain-specific parallel libraries (Quda, QDP++, Chroma, ...)
- Hand-tuned codes (MILC, ETMC, ...)
  - Many hand-tuned optimizations of formulas, using knowledge of *Dirac* and of preconditioning.
  - Many hand-tuned code optimizations achieved according to an **implicit** compiler/architecture and performance model.

Issues in code generation

- Parallelism not extracted automatically
- Changing algorithm, preconditioning $\rightarrow$ months of development effort
  - Adapt parallelism expression (MPI, OpenMP, accelerator and/or SIMD code)
  - Adapt data layout

Hampers algorithmic exploration

QIRAL: High Level Description for Lattice QCD

Three kinds of inputs:

- Definitions of constant matrices and vectors, such as the initial Dirac matrix
- Mathematical properties on matrices and vectors
- Algorithms and preconditionings used

QIRAL: High Level Description for Lattice QCD

Three kinds of inputs:

- Definitions of constant matrices and vectors, such as the initial Dirac matrix

$\rightarrow$ **Equations**

- Mathematical properties on matrices and vectors

$\rightarrow$ **Equations**

- Algorithms and preconditionings used

$\rightarrow$ **Rewriting rules**

Write equations, algorithms: `LaTeX`.

QIRAL: Expressions

Operations are a superset of SPIRAL:

- Usual vector/matrix operations, direct sum $\oplus$, tensor product $\otimes$

Predefined matrices

- Projection
- Some permutation matrices
- Identity matrix

$\oplus$ and $\otimes$ are building blocks for parallelism.

$$
\begin{aligned}
Dirac = {} & I_{L \otimes C \otimes S} \\
& + 2 * i * \kappa * \mu * I_{L \otimes C} \otimes \gamma_5 \\
& + \kappa * \sum_{d \in D} ((J_L^{-d} \otimes I_C) * \bigoplus_{s \in L} U(d)[s]) \otimes (I_S + \gamma[d]) \\
& + \kappa * \sum_{d \in D} ((J_L^{d} \otimes I_C) * \bigoplus_{s \in L} U(-d)[s]) \otimes (I_S - \gamma[d])
\end{aligned}
$$

QIRAL: Equations

Define properties on some matrices, specific to Lattice QCD:

$$
\begin{aligned}
U(d1)[s1]^{\dagger} &= U(-d1)[s1 + d1] \\
U(-d1)[s1]^{\dagger} &= U(d1)[s1 + -d1] \\
\gamma[d1]^{\dagger} &= \gamma[d1] \\
\text{diagonal}(\gamma_5) &= \mathit{true} \\
\gamma_5 * \gamma_5 &= I_S \\
\gamma_5 * \gamma[d1] &= -\gamma[d1] * \gamma_5 \\
\text{invertible}(I_S + c * \gamma_5) &= \mathit{true} \\
\text{invertible}(I_S - c * \gamma_5) &= \mathit{true}
\end{aligned}
$$

Domain specific properties for LQCD are all explicit

- QIRAL itself is not QCD dependent.

QIRAL: Algorithms and preconditionings

Algorithms as rewriting rules

- Replace some pattern (such as $x = A^{-1}b$) by a program, under some conditions
- Conditions checked by compiler.
- No need to be LQCD specific

Expressivity

- Iterative methods (CG, CGNR, GMRES, BICGSTAB, ...)
- Multi-precision methods
- Preconditionings (even-odd, inexact deflation in progress)
  - Easy to integrate variations around a given version

Template 1: Modified Conjugate Residual

Input : $A \in M, b \in V$
Output : $x \in V$
Match : $x = A^{-1} * b$
Require : diagonal$(A + A^{\dagger})$
Var : $r, r_1, Ap, p, Ar \in V, \alpha, \beta, n_r, n_{r1} \in \mathbb{C}$

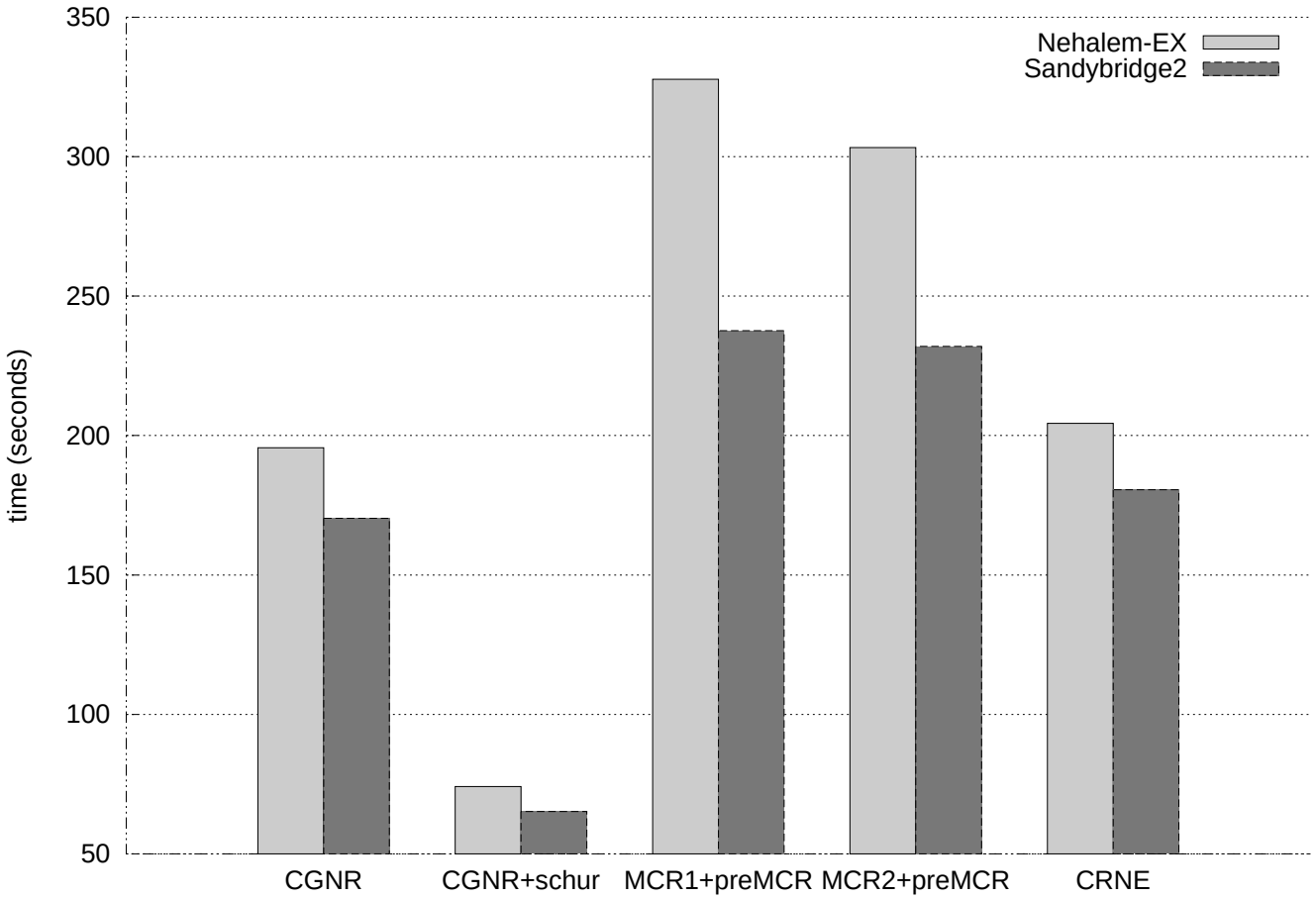
$r = b$;
$p = r$;
$x = 0$;
$n_r = (r \mid r)$;
$Ap = A * p$;
while $(n_r > \epsilon)$ **do**
  $\alpha = (Ap \mid r)/(Ap \mid Ap)$;
  $x = x + \alpha * p$;
  $r = r - \alpha * Ap$;
  $n_r = (r \mid r)$;
  $Ar = A * r$;
  $\beta = (Ap \mid Ar)/(Ap \mid Ap)$;
  $p = r - \beta * p$;
  $Ap = Ar - \beta * Ap$;

Code generation and optimization

Code Generation

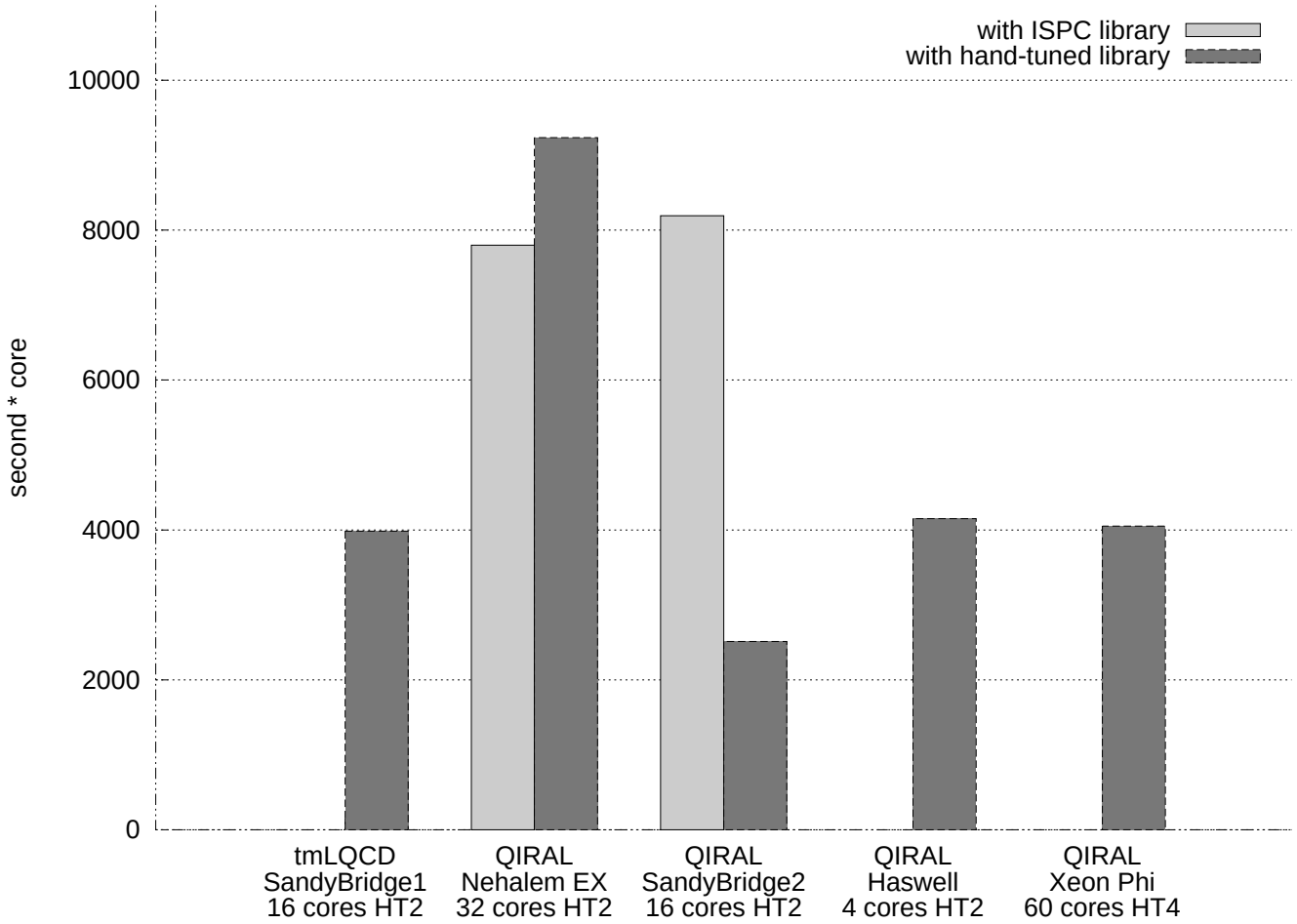
- Code as sequence/parallel loop around library calls (BLAS for instance)
- Code generated in C + pragmas.
- Easy to customize:
  - Specialized version of library calls
  - Vectorized, inline code dedicated to a given architecture
  - Independent of algorithmic part

Results: Comparing Methods



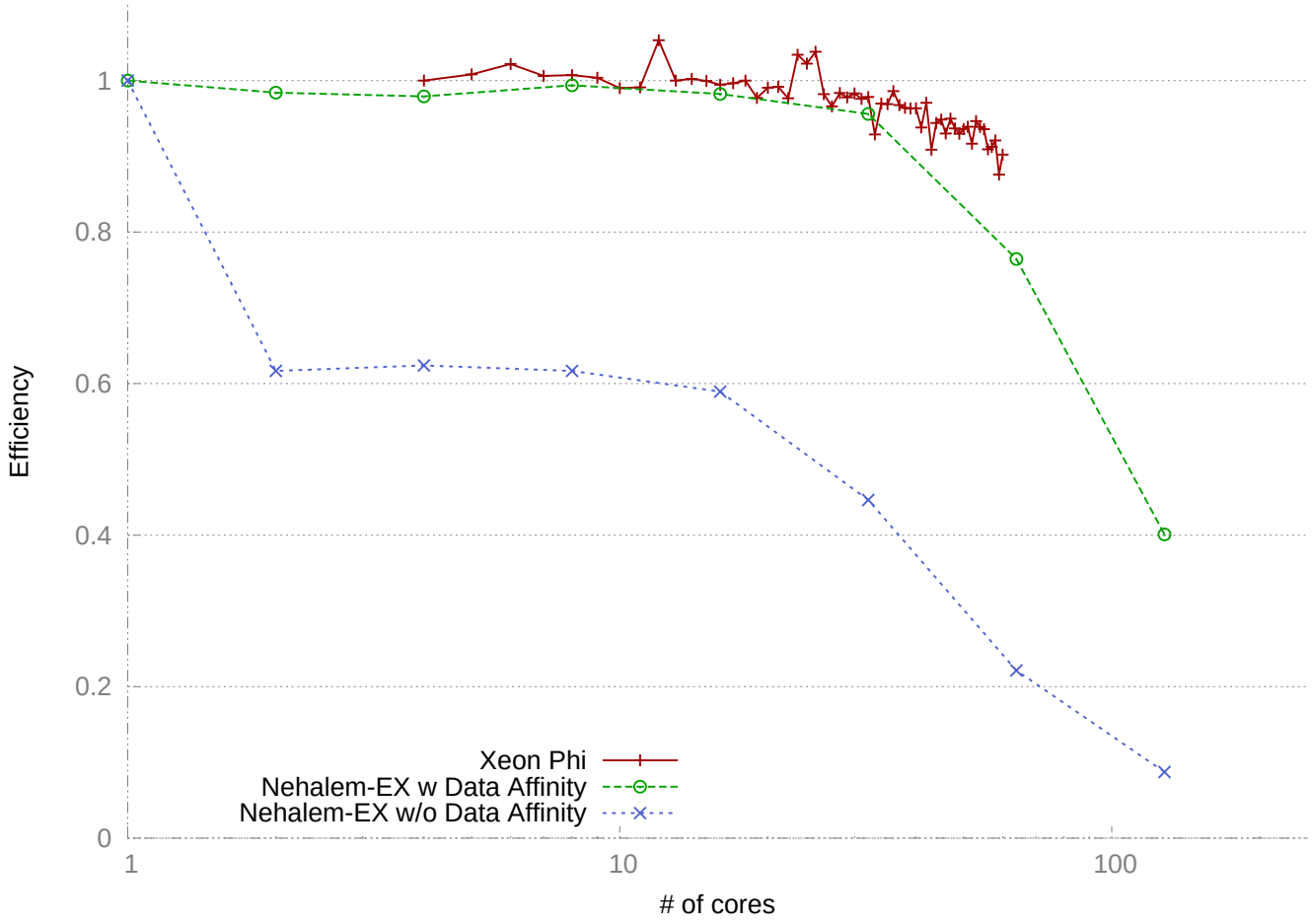
Lattice $24^4 \times 48$, error of 10^{-14}

Results: Same method, Comparing architectures



CGNR and even-odd, lattice $24^4 \times 48$, error of 10^{-14}

Results: Same method, Efficiency



CGNR and even-odd

Language for WP3, possibly WP6

Distributed computation and GPUs

- In progress
- Use of StarPU for heterogeneous architectures

Beyond LQCD

- Monte Carlo simulation of spin-glass systems
- Some Lattice Boltzman Simulation
- Shallow water/ AMR (add. effort needed)

Key for successfully using QIRAL

- Regular (cartesian) lattice