PROCEEDINGS OF SCIENCE

Lattice QCD performances on Aurora

Michele Brambilla
University of Parma and INFN
E-mail: michele.brambilla@fis.unipr.it

Francesco Di Renzo*
University of Parma and INFN
E-mail: francesco.direnzo@unipr.it

We present our most recent results for Lattice QCD performances on the Aurora parallel architecture. Aurora is based on Intel multicore CPUs and benefits from both standard (IB) and custom (3D torus) networks. Performances for LQCD are shown to effectively meet the expectations.

The 30 International Symposium on Lattice Field Theory - Lattice 2012,
June 24-29, 2012
Cairns, Australia

*Speaker.

PoS(Lattice 2012)192

1. Aim of the work

The key computational task of Lattice QCD is the application of the Dirac operator: the ubiquitous task in any of our programs is the iterative computation of its inverse. Here we will be concerned with the parallel computation of the so-called hopping term of the Wilson Dirac Operator (WDO)

$$\psi'_{\alpha i}(x) = \sum_{\mu=1}^{4} (U_{\mu}^{ij}(x)(1-\gamma_{\mu}^{\alpha\beta})\psi_{\beta j}(x+\hat{\mu}) + U_{\mu}^{ij}(x-\hat{\mu})(1+\gamma_{\mu}^{\alpha\beta})\psi_{\beta j}(x-\hat{\mu})) \tag{1.1}$$

All in all, making appropriate use of symmetries, this entails 1320 Floating Point (FP) operations on 360 FP words per lattice site.

The main issue for an efficient parallel computation (in our case, that of the WDO) is the quest for an optimal balance between computations and communications. Given a computational cost C to be assigned to N nodes (so that C/N is the computational cost per node), a computational performance P per node, a *local* exchange of information I/N, a memory bandwith B and *remote* exchange of information I_R and bandwith B_R, in first approximation one expects that the time T for the computation at hand equals [1]

$$T = \max\{\frac{C}{NP}, \frac{I}{NB}, \frac{I_R}{B_R}\} = \frac{C}{NP}\max\{1, \frac{IP}{CB}, \frac{I_R NP}{CB_R}\} \tag{1.2}$$

The previous formula gives a quantitative meaning to a simple expectation: if the computing time on each node equals the time required for data exchange, then there is a chance to hide communications with computations.

The platform on which we performed our numerical benchmarks is the Aurora system that the AuroraScience collaboration [2] has been running for a couple of years [3]. The machine is a highly dense, liquid cooled parallel system, based on Intel multi-core Xeon CPUs and endowed with both a commodity IB network and a custom TORUS [4] network . AuroraScience has been an FBK/INFN joint initiative, in collaboration with Eurotech. The present work builds on experience we reported on in [5].

2. The quest for optimally balancing computations and communications

In recent years a common trend has been the increase in the number of basic processing units on a single node. This is becoming more and more extreme in devices such as Graphics Processing Units (GPUs) used for General-Purpose computations (the so-called GPGPU paradigm) or in many-cores CPUs (at the moment mainly the Intel MIC). This trend forces us to confront a problem that has already shown up for systems like Aurora (based on multi-cores CPUs). We are talking of the interplay between the intra-node and inter-nodes levels of parallelism.

2.1 Intra-node parallelism

Eq. (1.2) generically refers to *remote* bandwith and exchange of informations. This should be contrasted with *on-node* bandwith and exchange of informations. In a nutshell, the main issue

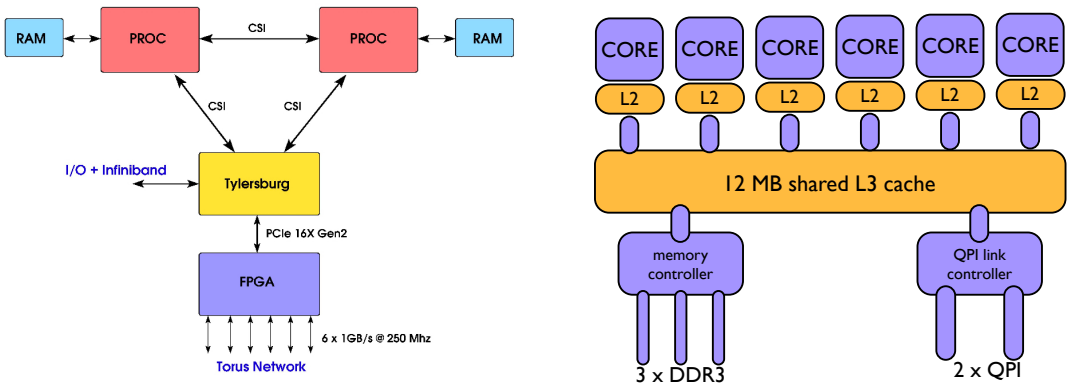

Figure 1: The Aurora nodecard (left). Basic structure of the Intel Westmere processor (right).

of the interplay between the intra-node and inter-nodes levels of parallelism can be phrased in a simple question: what is a node on nowadays systems?

In Fig. 1 we show both the basic architecture of the Aurora nodecard and a sketch of the Intel Westmere processor (that is the one our system is based on). On the left one can see that an Aurora node is an SMP dual-processor system, which is connected to remote nodecards either via a standard IB network or via a custom Torus network (whose firmware resides on an FPGA). One has a remote communication each time either IB or the Torus is accessed. However, it is trivial to observe that without any data exchange taking place via either network, there is possibly data exchange between the two CPUs via the so-called QPI link. On the right one can see that the 6 cores on a single CPU share an L3 cache memory. They access the DDR3 memory of their own socket via the memory controller and (as alredy pointed out) the DDR3 memory attached to the other socket via a QPI link controller. One can conclude that, strictly speaking, there is double layer of parallelism even inside each nodecard.

A lattice is partitioned into sub-lattices resinding on each node. Roughy speaking, it is the optimization of such a partitioning that will eventually decide for efficiency. Our main choices are the following:

- Our parallel program is an MPI/multithread one. A single MPI rank resides on each nodecard and 24 threads are at work within it. This means that we make use of HyperThreading (HT), a choice that will be further commented later.
- To benefit from SSE instructions we explicitly code them via the so-called *SSE intrinsic* instructions.
- Any attempt to optimize memory usage (and even L1/L2/L3 cache memory usage) requires to go through *core affinity* setting and *memory binding*. The former enables the assignment to a given core of a given task (typically, taking care of a fraction of the sub-lattice which is assigned to the node). This is even more important given that we make use of HT (we will further comment on this). With *memory binding* one attaches memory to a given socket.

lattice size	GFlops
$8 \times 4 \times 24^2$	52.6
$8 \times 12 \times 48^2$	37.7
$12^2 \times 48^2$	30.4
$12 \times 24 \times 48^2$	26.8
$12 \times 48^2 \times 96$	22.5

Table 1: Single node performances of the Wilson Dirac Operator benchmark for various lattice sizes. Peak performance of an Aurora nodecard is 160 GFlops.

This has mainly to do with the allocation of the (sub)lattice within the node: with a given choice of the slowest-running direction, it is easy to split both threads and data among the two sockets *along that direction*. As a result, most of the time each thread will be operating on data allocated on the same socket the thread itself is attached to.

These three choices basically control the intra-node efficiency. To quantify how well we are doing at this (intra-node) level of parallelism, we run simulations on a single node (*i.e.* we have no inter-nodes communications), with a choice of lattice sizes that would make sense in real, parallel simulations. Our intra-node results (with various choices of sizes) are reported in Tab. 1 (one can easily spot cache effects). We quote performances on a single node; peak performance is around 160 GFlops. All the performances we refer to are in Double Precision.

2.2 Inter-nodes parallelism

Before we describe the parallel features of our code, we make a few basic comments on the general problem of inter-nodes parallelism. Taking a very pragmatic attitude, we can say that two main issues are the partitioning of the lattice into sublattices and the choice of data layout. In the end one looks for appropriate choices being aware of the following:

- One wants to hide communications by overlapping them with computations.
- A data layout that is efficient for intra-node parallelism is not necessarily ready for remote data exchange (actually, neither necessarily ready, nor efficient).

MPI provides solutions taking care of both issues:

- One makes use of *non-blocking* communications routines.
- For most of the choices made with respect to data layout, MPI can *pack and unpack* data.

With respect to the second issue, it is well known that changing data layout can result in sizeable differences. In Fig. 2 we show different performances of the ETM Collaboration code [6] (again, we look at the WDO benchmark). Programs were run on the same Aurora system, with different data layout [2]. Fig. 2 depicts so-called *strong scaling*: a given lattice size is partitioned among an increasing number of nodes.

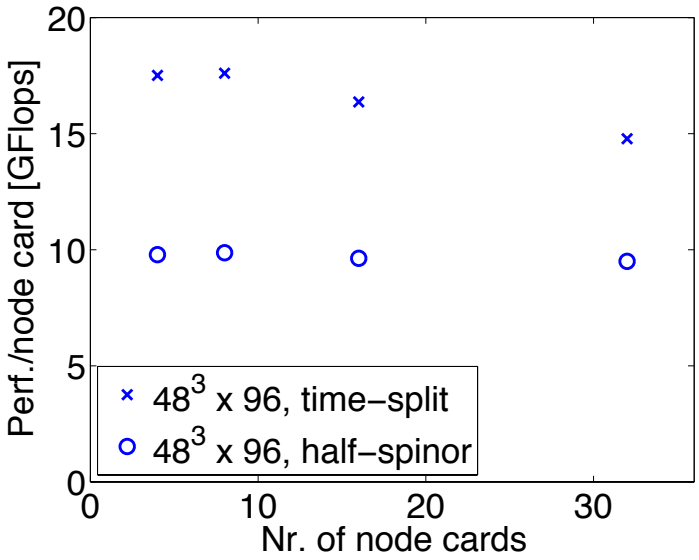

Figure 2: Performances of the ETMC code with different data-layouts.

3. Tailoring a code to the Torus network

We now sketch the basic strategy of our WDO benchmark with respect to inter-nodes parallelism. The choices aim to exploit the custom Torus network.

- We directly manage the buffers for data exchange. These need to be aligned and (like any other data) are allocated making use of memory-binding.

- Hyperthreading is in place. In particular, to overlap computations and communications, we explicitly have at some point threads performing computations and threads performing communications. It is known that in general HT can give a rather modest performance increase: not surprisingly, two threads on the same physical core do not perform as two physical cores. By managing core-affinity, we make sure that on each core there is one thread performing computations and one thread performing communications: since competition on FP resources is thus negligible, this choice results in rather good efficiency. We also point out that thread allocation changes during the execution of different tasks.

- We make use of symmetries and only send and receive *half-spinors*. More explicitly, spinors are projected while they are arranged on data-sending buffers and reconstructed while they are fetched from data-receiving buffers. We also make use of *even/odd* sites independent update.

To better understand a few more details of the code one can refer to Fig. 3, which is a cartoon for our communication strategy. Borders are color-coded in order to distinguish *local* borders (orange) and *remote* borders (blue). According to the same color-code, data-sending buffers are orange, data-receiving buffers are blue. It is useful to look at Eq. (1.1) itself in a color-coded form.

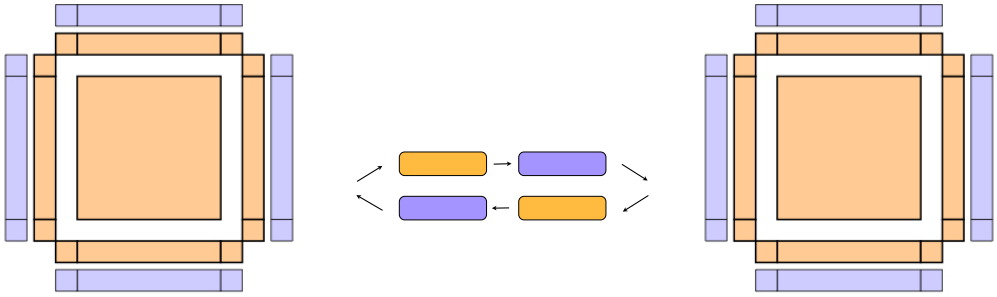

Figure 3: A cartoon for our communication strategy.

$$\psi'_{\alpha i}(x) = \sum_{\mu=1}^{4} (U_{\mu}^{ij}(x)(1-\gamma_{\mu}^{\alpha\beta})\psi_{\beta j}(x+\hat{\mu}) + U_{\mu}^{ij}(x-\hat{\mu})(1+\gamma_{\mu}^{\alpha\beta})\psi_{\beta j}(x-\hat{\mu}))$$

More explicitly, the program runs as follows:

- In a first (rather quick) phase, all the threads work together in preparing the data-sending (orange) buffers. In this phase the most relevant part of the computation is the one in orange, in which one computes $U_{\mu}(x-\hat{\mu})\psi(x-\hat{\mu})$, then projects the half-spinor and places it on the data-sending buffer.
- In the following phase half of the threads update the bulk of the (sub)lattice (no remote communication is needed; these are the computation threads), while half of the threads take care of the communications exchanging the borders. Notice that defining what is bulk is a bit non-trivial (depending in particular on the direction μ).
- Finally, all the threads work together in reconstructing the data that are fetched from the data-receiving (blue) buffers. In this phase the most relevant part of the computation is the one in blue, in which one fetches the half-spinor from the data-receiving buffers, then computes $U_{\mu}(x)\psi(x+\hat{\mu})$, reconstruct the whole spinor and places it into its position.

Perfomances figure can be read in Fig. 4, where we compare them with those of the (standard) ETMC code (actually, the version with the data layout getting better performances on Aurora); the latter code relies on the IB network.

It would be interesting to test how much one can improve by exploiting the AVX instruction set, introduced on the SandyBridge CPU. Even more interesting will be to investigate how much of the expertize gained on multi-core architectures (core-affinity, memory binding) can be effectively exploited on many-cores architectures (namely, on the Intel MIC).

Acknowledgments

The AuroraScience project has been funded by the Provincia Autonoma di Trento and the Istituto Nazionale di Fisica Nucleare, in the framework of an agreement with the Fondazione Bruno

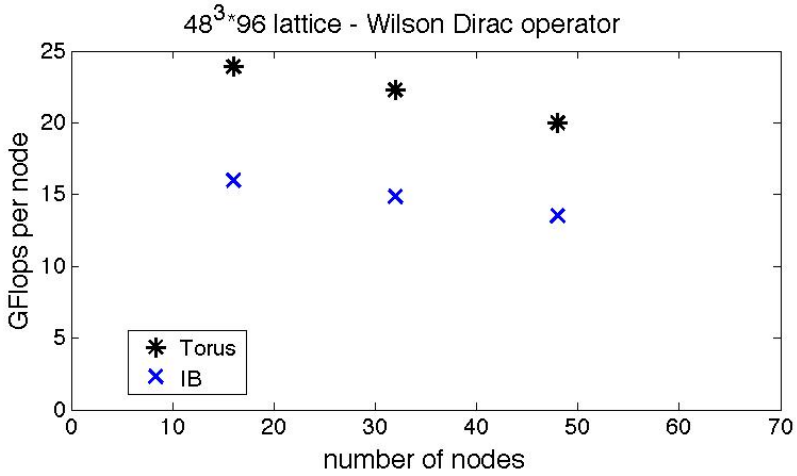

Figure 4: Performances of the code based on Torus-network communications vs the one making use of IB communications.

Kessler. Lattice QCD software developments and applications on Aurora are also supported by ITN STRONGnet (European Union Grant Agreement PITN-GA-2009-238353). We thank all the members of the AuroraScience collaboration for useful discussions.

References

[1] For a rigorous presentation see G. Bilardi et al., *The Potential of On-Chip Multiprocessing for QCD Machines*, Springer Lecture Notes in Computer Science 3769 (2005) 386.

[2] L. Scorzato, *AuroraScience*, PoS (Lattice2010) 039.

[3] F. Di Renzo, *Status of the AuroraScience project*, PoS (Lattice2011) 031.

[4] M. Pivanti, F. Schifano and H. Simma, *An FPGA-based Torus Communication Network*, PoS (Lattice2010) 038.

[5] M. Brambilla, F. Di Renzo, M. Grossi, *Efficiency on multi-core CPUs: the Wilson Dirac operator on Aurora*, PoS (Lattice2011) 302.

[6] K. Jansen and C. Urbach, *tmLQCD: A Program suite to simulate Wilson Twisted mass Lattice QCD*, Comput. Phys. Commun. 180 (2009) 2717.