Contents lists available at ScienceDirect

Journal of Computational Science

journal homepage: www.elsevier.com/locate/jocs

Unite and conquer approach for high scale numerical computing

Nahid Emad[a,b,*], Serge Petiton[a,c]

[a] Maison de la Simulation, Digitéo Labs – Bat. 565, 91191 Gif-sur-Yvette Cedex, France
[b] LI-PaRAD Laboratory, 45 avenue des Etats-Unis, 78035 Versailles Cedex, France
[c] CRIStAL, University Lille 1, Sciences and Technologies, France

ARTICLE INFO

Article history:
Received 15 August 2015
Received in revised form
27 December 2015
Accepted 24 January 2016
Available online 6 February 2016

Keywords:
Restarted hybrid methods
Krylov methods
Linear algebra
Parallel and distributed programming model
Extreme scale computing

ABSTRACT

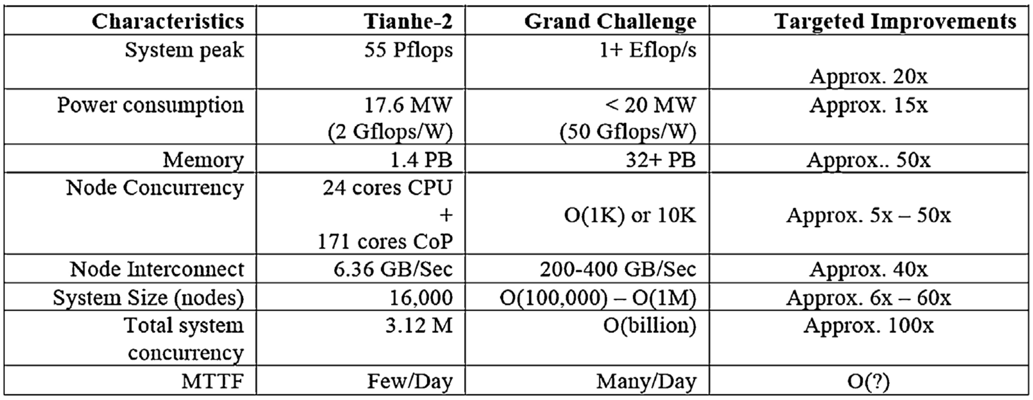
The ability to exploit emerging exascale computational systems will require a careful review and redesign of core numerical algorithms and their implementations to fully exploit multiple levels of concurrency, hierarchical memory structures and heterogeneous processing units that will become available in these computational platforms. This paper presents the “unite and conquer” approach to solve linear systems of equations and eigenvalue problems for extreme scale computing. Indeed, there are two ways to optimize the execution of a restarted method on a large-scale distributed system. The first one is to optimize the number of floating point operations per restart cycle through maximizing the concurrency inside a restart cycle while minimizing latencies. The second way is to accelerate/improve the rate of convergence for a given computational scheme. The unite and conquer restarted approach focuses on decreasing the number of restart cycles by coupling either synchronously or asynchronously several restarted methods called also co-methods. In the end of a restart cycle, each co-method locally gathers available results of all collaborating co-methods and selects the best one in order to create its restarting information. Consequently this permits the global reduction of the number of cycles to convergence. The unite and conquer restarted methods are heterogeneous, fault tolerant, support asynchronous communications and present a big potential of load balancing. Due to these properties, they are well adapted to large-scale multi-level parallel architectures. We show the relevant programming paradigms that allow multi-level parallel expression of these methods and how the software engineering technology can contribute significantly in achieving high scalability. We present some experiments validating the approach for unite and conquer restarted Krylov methods on several parallel and distributed platforms.

1. Introduction

In the coming years, the number of cores along with heterogeneity of components of supercomputers (such as GPUs and other accelerators) will grow causing the transition to multi- and many-cores inside computing nodes that communicate explicitly through fast interconnection networks. These very hierarchical supercomputers will be at the intersection of distributed and parallel computing and pose big challenges that have to be addressed (see Fig. 1) [1]. The communication, energy consumption, fault tolerance are becoming increasingly important inside such systems. To harness the totality of the computational power in such a complex ecosystem, it is central to explore novel parallel programming and execution models, which put the emphasis on multi-grain multi-threading, multi-level memory and parallel processing, reducing synchronizations and promoting asynchronicity, multi-level scheduling strategies, etc. The complexity of these architectures and recent and emerging prototypes must be taken into account to design future languages and frameworks and anticipate their characteristics, by proposing new programming paradigms, new general purpose as well as application-oriented languages and environments (DSL). Indeed, the SPMD/MPI-Like models will not be sufficient for architectures with millions of cores and billions of threads. Taking into consideration the characteristics of these supercomputers may help to define sets of constraints to be satisfied in order to improve the performance of large applications on these systems. Based on these constraints, we can then define smart co-design methods/algorithms associated with these applications. Systems and languages are often defined on the basis of existing methods, algorithms and libraries but they were developed only for SPMD and MPI-like programming models and the time when Moor’s law was verified. But for new computational systems, numerical methods have to be adapted

* Corresponding author at: Maison de la Simulation, Digitéo Labs – Bat. 565, 91191 Gif-sur-Yvette Cedex, France.
E-mail address: Nahid.Emad@uvsq.fr (N. Emad).

http://dx.doi.org/10.1016/j.jocs.2016.01.007
1877-7503/© 2016 Elsevier B.V. All rights reserved.



Characteristics	Tianhe-2	Grand Challenge	Targeted Improvements
System peak	55 Pflops	1+ Eflop/s	Approx. 20x
Power consumption	17.6 MW (2 Gflops/W)	< 20 MW (50 Gflops/W)	Approx. 15x
Memory	1.4 PB	32+ PB	Approx.. 50x
Node Concurrency	24 cores CPU + 171 cores CoP	O(1K) or 10K	Approx. 5x – 50x
Node Interconnect	6.36 GB/Sec	200-400 GB/Sec	Approx. 40x
System Size (nodes)	16,000	O(100,000) – O(1M)	Approx. 6x – 60x
Total system concurrency	3.12 M	O(billion)	Approx. 100x
MTTF	Few/Day	Many/Day	O(?)

Fig. 1. Key characteristics and challenges of forthcoming exascale architectures.

by taking into account the architecture, arithmetic, I/O latencies, etc. In this context, auto-tuning has to become a general approach. Some numerical method has to be coupled to solve large scientific applications asynchronously with each collaborating method (co-method) auto-adapted. Fine criteria at the application level and/or mathematical methods level should be taken into account in auto-tuning resulting in "smart-tuned" algorithms. The correlation between these elements gives rise to intelligent numerical methods for which "end-users" will be able to provide their expertise. Our goal is to discuss the trend of parallel programming and execution paradigm for high scale parallel/distributed computing and to show the adaptability of it to the case of modern smart-tuned numerical methods. For these reasons, we propose "unite and conquer" approach and, focus then on the case of unite and conquer restarted Krylov subspace methods.

In the following section, we present key elements for programming paradigms of targeted computing systems. A programming model is proposed together with corresponding YML framework offering the required properties necessary for its implementation. Section 3 describes the unite and conquer approach and its application to iterative methods, leading to design and deployment of the hybrid methods. Some experimental results pointing out the efficiency of the approach are presented. Section 4 focuses on the multiple restarted Krylov subspace methods, a particular case of hybrid ones, and their performance on various parallel and distributed platforms. Concluding remarks and perspective are presented in Section 5 including the highlighting of some unexpected aspects of these hybrid methods in the case of multiple restarted Arnoldi method with nested subspaces (MRAMns).

2. What programming paradigms?

In order to provide targeted programming models, the following key elements have to be considered. *Data movements* have to be minimized, but it must be remembered that all communications do not have the same cost. The latencies between distant cores are expected to be very time-consuming. Consequently, communications between distant cores have to be avoided thus reducing the *energy consumption* of these systems. Programming and execution models have to consider *multi-level parallelism*, from very coarse-grained through medium grained down to fine-grained parallelism with *scheduling* strategies, *load balancing* policies and *fault tolerance* in each level.

2.1. Our proposition

We propose a programming paradigm suitable for these large parallel and distributed architectures which is based on the following two factors.

(1) Component approach. The targeted high performance computing systems have a large number of nodes and cores with a part of them being geographically apart. Additionally, the heterogeneity of their basic elements (processors, memories, run-time systems) makes them very complex architectures. Most current applications and numerical libraries neither efficiently utilize the number of cores per node nor the heterogeneity of the processing units in the systems. This clearly could result in poor resource utilization that keeps computational scientists looking for short-term solutions that in the long run becomes very costly. Furthermore, the basic constituents of these parallel applications and libraries are strongly intertwined. That means, the computations, the data and the communication are often defined in the same program entity, preventing the extensibility and interoperability and as a consequence the reusability of the applications.

In order to effectively exploit these architectures and design sustainable programs, it is essential to be able to define an application as a set of off-the shelf and communicating components. Thereby, design and implementation of programming models for such complex systems has to be governed by component-oriented approach. Our oncoming is based on an innovation oriented-component approach based on the abstraction of the three main elements of a parallel numerical program which are computation, data management and communication [2]. In addition to other good software engineering consequences such as interoperability, extensibility, and durability, this will allow to achieve the code reusability in terms of software as well as hardware. The first provides the possibility of making use of the same code on several types of architectures such as vector ones, accelerators, etc. The latter permits, inter alia, to make use of the same code in several different contexts such as numerical methods, visualization, data managements, etc. As a consequence, in addition to design of reusable applications and/or libraries, this programming model allows making (re)use of existing efficient libraries in the context of extreme scale computing. For that, these libraries have to be equipped with software mechanisms that will enable us to deploy different parallel programming constructs with minimal changes to their current user interfaces.

(2) Graph of components. Once the component approach is adopted, we propose to describe an application as graph of its coarse grain components. Such components can themselves be a graph of tasks and can be described by a data-flow, SPMD PGAS[1]-like or by a data-parallel programming model. Thus, on each processor or node hosting such a component, we can program accelerators and on the level of cores, multithreaded optimization and runtime libraries can be used. This programming model

[1] Partitioned Global Address Space.