Technical Report

TR-2012-004

Experiences with Target-Platform Heterogeneity in Clouds, Grids, and On-Premises Resources

by

Jaroslaw Slawinski, Tiziano Passerini, Umberto Villa, Alessandro Veneziani, Vaidy Sunderam

Mathematics and Computer Science

EMORY UNIVERSITY

Experiences with Target-Platform Heterogeneity in Clouds, Grids, and On-Premises Resources

Jaroslaw Slawinski, Tiziano Passerini, Umberto Villa, Alessandro Veneziani, Vaidy Sunderam
Mathematics & Computer Science, Emory University, Atlanta, USA
{*jslawin, tpasser, uvilla, avenez2, vss*}*@emory.edu*

***Abstract*—Heterogeneity in secondary characteristics of different HPC target platforms is the focus of this paper. Clusters, grids, and (IaaS) clouds may appear straightforward to configure to be interchangeable – but our experiences with mainstream parallel codes for CFD demonstrate that secondary attributes – support software, interconnect type, availability, access, and cost – expose heterogeneous aspects that impact overall effectiveness of application execution. The emergence of clouds as alternatives to grids and local resources for parallel HPC codes portends "computing as a utility" in science and engineering domains. Our experiences provide preliminary insights into characterizing these different types of platforms to which users typically have access – and show where the tradeoffs can be, in terms of deployment effort, actual and nominal costs, application performance, and availability (both in terms of resource size and time to gain access). For our test application, we report that each of the platforms to which we had access had its particular benefits and drawbacks in terms of the above attributes. More generally, our experiences may provide an example preview into what developers and users can expect when selecting a "utility provider" and specific instance thereof for a particular run of their application.**

***Keywords*-Platform heterogeneity; Cloud computing; Scientific applications; Cost characterization;**

I. Introduction

Computing as a utility has become a reality in many domains; Clouds deliver storage and processing resources on demand via methods analogous to more traditional utilities. Such a paradigm is evolving for high performance science and engineering applications (*High Performance Computing* – HPC). Typically, applications in the HPC domain are characterized by computing and/or data intensive codes that are parallelized explicitly, commonly based on the message passing programming model. These applications largely execute on local, on-premise clusters or on platforms referred to as computational grids – although in practice, grid-computing predominantly manifests simply as remote access to clusters, just in a different administrative domain. In both settings, it has been traditional to measure the performance of HPC applications by a single metric *viz.* time to completion for the particular application in question, parameterized along two dimensions: problem size and number of processing elements used. With the advent of cloud computing, two interesting perspectives have become relevant: (1) the viability of executing parallel applications on the cloud (either through self-assembly or renting a pre-built cluster); and (2) the actual dollar cost effectiveness of executing HPC applications on different target platforms. In this paper we report on preliminary experiences with executing a Finite Element Method (FEM) code on four different platforms that are heterogeneous in secondary respects (interconnect, access method, use cost) and attempt to characterize the overall "expense factor" of each. We provide some background information on normal modes of scientific application execution and subsequently outline the FEM code used in our exercise. We then describe the process and issues involved in preparing and deploying the application on four different platforms. Measurements of execution time, augmented with usage cost and (qualitative) deployment effort are presented and discussed; the paper concludes with a summary of factors that characterize the effectiveness of using different kinds of platforms.

II. Background

HPC is intrinsic and integral to most fields of scientific endeavor. Message passing parallel programs are a staple modality of numerical simulations and computational analyses. In addition to the parallel framework, e.g. MPI, codes depend on various other auxiliary components: scientific and mathematical libraries, header files, particular compiler options and flags. These parameters (or sets thereof) are quite specific to a particular *target platform*; executing the application on a different target platform may require a non-trivial amount of re-building effort (even if the actual application source code is untouched). Hence, applications continue to be executed on the default "home" platform even if other viable options are present.

Grids and especially Clouds present real opportunities for applications to move away from their home environments. If an application run can be obtained in minutes on the Cloud instead of waiting for overnight turnaround times on a local cluster, clouds may be an attractive proposition – provided the monetary and manpower costs are acceptable [1]. In the ADAPT project at Emory, we are investigating the feasibility and ease of deploying classes of applications on target platforms other than those on which they normally execute. As a learning exercise, we have experimented with a Finite Element Method (FEM) CFD code based on the C++ library

LifeV [2], whose home environment is an 128-core cluster, and ported it on other computational platforms: clusters and Amazon's Cloud.

III. Related work

The role of cloud computing as an extension of current HPC capabilities has been evaluated by many researchers. In various scientific fields, the rate of increase of available computing power is closely matched or outpaced by the increase in model complexity and therefore of the requirements for fast, large scale computations – prompting serious consideration of "unlimited, on-demand resources" that clouds promise. This however is still controversial [1], [3], [4], [5]. Cloud vendors have been reshaping their services, experimenting with new technologies, and exploring new price policies while users are assessing viability. Several cloud-effectiveness benchmarks have appeared in the literature ([3], [6]). We believe, however, that an assessment of cloud computing as a viable choice in real-life applications requires evaluation of its support for more complex scientific software, as we detail in the next section. The present work also includes early benchmarks of Amazon's `cc2.8xlarge` instances, a novel computational offering that is a candidate to match the performance of traditional computing clusters. Furthermore, most studies focus on time-to-completion; our study takes a broader perspective, including a preliminary assessment of cost aspects [7], and the set of activities required to prepare the execution environment for scientific codes on diverse platforms.

The use of the Cloud as the computational platform for computational fluid dynamics analysis has been explored by several software projects. Among the open source projects we cite CAELinux [8], a Linux distribution including a large set of open source packages for computer-aided engineering (Salome (Open CASCADE) [9], Code_Aster (EDF) [10], Code_Saturne (EDF) [11], OpenFOAM (SGI Corp) [12] and Elmer (CSC) [13]). CAELinux currently supports cloud execution on Amazon EC2 by providing a set of pre-defined virtual machines to be run on the EC2 service. OpenFOAM, an open source package for CFD analysis, can be also executed as a standalone package on the Amazon EC2 [14] computing service and on the SGI Cyclone Technical Computing Cloud. We note here that our work is concerned with *comparing* effort, cost, and issues in executing applications on *multiple target platforms exhibiting secondary heterogeneity* rather than the aspect of porting applications to the cloud.

IV. The numerical problem and its solver

Partial differential equations (PDE) are a formidable tool for modeling problems in different fields, ranging from aerospace and automotive, mechanical and structural engineering to biology and biomedicine, ecology and finance [15]. Explicit and analytical solutions to PDE's of real interest are seldom available and numerical approximations are the norm [16]. FEM is a well established approach to the numerical solution of PDE's [17], [18]. The FEM solution is a piecewise polynomial approximation of the exact one and the differential problem is replaced by an algebraic (linear) system. The accuracy of the approximation is in general related to the size of each portion ("element") of the computational domain where the solution is assumed to be polynomial. The finer the reticulation (*mesh*) defining the elements, the larger the algebraic problem to be solved after discretization – and consequently, the computational cost – but the more precise the solution.

A. First test case: reaction-diffusion equation

As a first simple test case, in this paper we consider the following PDE in a cubic region

$$\frac{\partial u}{\partial t} - \frac{1}{t^2}\sum_{i=1}^{3}\frac{\partial^2 u}{\partial x_i^2} - \frac{2}{t}u = -6. \qquad (1)$$

Boundary and initial conditions are selected in such a way that the exact solution is $u = t^2(x_1^2+x_2^2+x_3^2)$ (figure 1). This is generally called a *Reaction-Diffusion* (RD) equation. More details about this test case can be found in [16], Chap. 5. Exact solution is used for checking the mathematical correctness of the code execution.

Since the unknown u in equation (1) depends on time t and on the space coordinates x_i, the numerical solution requires both time and space discretization. We use a second order Backward Difference Formula (BDF) for the time derivative and the FEM of order 2 for the space variables. In particular, we use the research C++ library LifeV, developed as a joint project among the Departments of Mathematics at the EPFL, Lausanne, Switzerland, and the Politecnico di Milano, Italy, the INRIA in Paris, and the Department of Mathematics and Computer Science at Emory University. The library has been mostly developed for applications of the FEM in blood flow and industrial problems.

B. Second test case: incompressible Navier-Stokes equations

Incompressible fluid dynamics represents one of the most challenging, attractive and impactive problems in modern scientific computing. Fast and reliable numerical solutions of the Navier-Stokes equations (NSE) – the basic mathematical model for incompressible fluid dynamics – are required in several engineering fields, ranging from automotive/aerospace to geophysical and biomedical engineering [19], [20]. If $[u_1, u_2, u_3]$ denotes the velocity vector and p the pressure of a liquid in the 3D space with coordinates

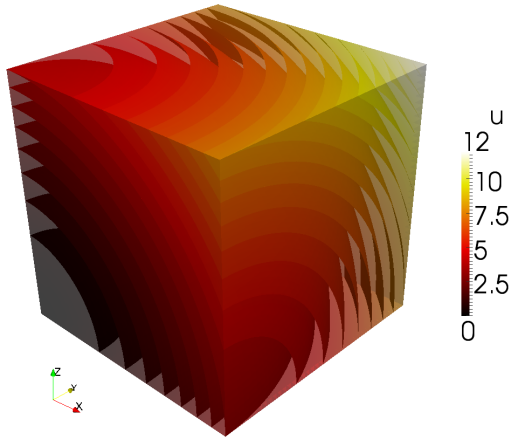

Figure 1. Solution of equation (1) when $t = 2$s. The isosurfaces of u are plotted inside the cubic domain, for a set of 25 values chosen with a constant interval of 0.5

x_1, x_2, x_3, the incompressible NSE read

$$\rho \frac{\partial u_i}{\partial t} - \sum_{j=1}^{3} \left(\frac{\partial}{\partial x_j} \mu \left(\frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i} \right) + \rho u_j \frac{\partial u_i}{\partial x_j} \right) + \frac{\partial p}{\partial x_i} = f_i \quad i = 1, 2, 3, \tag{2}$$

$$\sum_{j=1}^{3} \frac{\partial u_j}{\partial x_j} = 0.$$

Here ρ is the fluid density and μ is the viscosity (that we assume to be constant for simplicity). Vector $[f_1, f_2, f_3]$ is an external forcing term. From the numerical viewpoint, this problem is by far more challenging than RD equation (1), not only due to size (this is a vector problem involving four scalar fields), but owing to intrinsic mathematical features and the non-linear term (see, e.g. [20]). In this paper, in particular, we use for our experiments a classical problem proposed by C. R. Ethier and D. A. Steinman [21], a popular non-trivial benchmark for CFD solvers. The time derivative is discretized with a second order BDF, while the unknowns u and p are approximated using the FEM of order 2 and of order 1 respectively. The exact solution of this problem is shown in figure 2.

C. The organization of the program

The numerical solution of problems like the proposed test cases involves operations that are conceptually split into two categories. The evolution in time is solved as a sequence of steps that compute the unknowns at selected instants t^k. Some operations are independent of the time advancing and are performed out of the temporal loop. Other operations need to be performed at each time step. These typically constitute the computationally-intensive kernel of

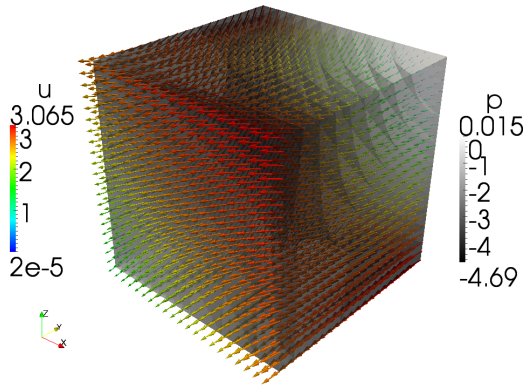

Figure 2. Solution of the problem proposed by C. R. Ethier and D. A. Steinman [21], based on equation (2), when $t = 0.003$s. Arrows represent the vector field u, while in the cubic domain are shown isosurfaces of the scalar field p

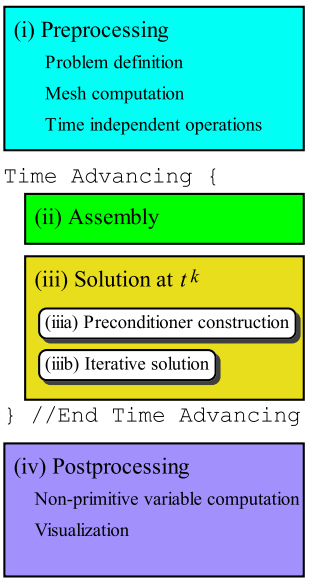

Figure 3. Steps for the numerical solution of a time-dependent PDE problem

the software. Schematically, we represent the stages of the application as in figure 3.

Here we detail each phase. Step (i) consists of the definition of the *computational domain* where the equations have to be solved numerically. This is given by the mesh. This task is typically accomplished with in-house mesh generators (for structured meshes) or third-party software such as NetGen [22] and GMSH [23]. In a parallel application, the domain is *partitioned* so that each process takes care only of a subset of the global mesh. This splitting is achieved by resorting to graph partitioning algorithms, such as those implemented in the library ParMETIS [24], guaranteeing a proper load balancing among processes. The

load is measured as the number of mesh elements assigned to each process. Other operations of this step refer to all the computations that are time independent and can be performed once.

Step (ii) concerns the computation (or more specifically the *assembly*) of the matrices and vectors required for the construction of the discretized algebraic problem. This is carried out with algorithms and data structures provided by LifeV. In a parallel application, each process can only access a subset of the matrices and vectors, corresponding to its own portion of the mesh. In other terms, matrices and vectors are distributed and need to be updated via a message passing interface. Our software uses distributed data structures implemented in the external library Trilinos [25] by Sandia Labs. Trilinos also provides algorithms for the solution of the algebraic problem (step (iii)). In particular, we use *iterative preconditioned methods*, where the solution of the large linear systems assembled at each time step is replaced by the iterative solution of simpler systems (called *preconditioners*). For this reason, we distinguish a step (iiia) for the computation of the preconditioner, and step (iiib) for the actual solution of the preconditioned system.

Step (iv) concerns the visualization of the solution to the differential problem, and can be delegated to third party software such as Paraview [26]. This step may also include the computation of quantities of interested related to the solution u.

For the purpose of the present paper, we are mostly concerned with steps (ii) and (iii), which have a major impact on the entire computational cost of the application.

D. Summary of the packages used in LifeV

The complete list of required packages to build our PDE solver follows:

- LifeV library [2], for the formulation of the algebraic counterparts to differential problems; this library is used directly by the solver application;
- Third-party scientific libraries:
  - Trilinos [25] for the solution of linear systems (data structures and algorithms);
  - ParMETIS [24], used for mesh partitioning;
  - SuiteSparse [27], as a support library extending the capabilities of Trilinos;
  - BLAS/LAPACK libraries (generic or vendor-specific implementations);
- General-purpose and communication libraries:
  - Boost C++ libraries [28], mainly used for effective memory management (smart pointers);
  - HDF5 [29], for the storage of large data on file. For compatibility issues, this package has to be built with the 1.6 version interface;
  - MPI libraries (e.g., Open MPI);
- Compilers:
  - C++ compiler (e.g., GCC version 4 or above);
  - [optional] Fortran compiler, compatible with C++;
- Deployment tools:
  - GNU make;
  - Autotools;
  - CMake (version 2.8 or above).

V. Four Heterogeneous Target Platforms

We benchmarked two applications for the numerical solution of the two test cases presented in the previous section, on four different computing architectures.

As the starting point for our analyses, we selected an in-house computing cluster constituting a computational test bed for the LifeV developer team. [1] As the second architecture, we used a larger compute cluster provided on a fee-for-use basis within our university. Next, we evaluated the usability of on-demand resources provided by Amazon's Elastic Compute Cloud (EC2). From the rich resource offerings provided by this vendor, we picked the most powerful hosts, dubbed *Cluster Compute*. The fourth platform was the HPC supercomputer available for scientists at the CILEA supercomputing center, in Segrate (Milano), Italy – this exemplifies canonical grid usage.

The four platforms are heterogeneous in many respects: they differ in availability (measured as wait time to obtain access to the machine), access modality (privileged vs. unprivileged user), storage (e.g., size of user disk space), build (e.g., presence of the compilers and basic build tools), aggregation (e.g., presence of MPI toolsets), and execution (e.g., presence and type of parallel job schedulers). In this section we compare the considered architectures, pointing out differences and similarities. Table I summarizes the compared features; below we note a few relevant details.

A. Puma

The in-house computing cluster `puma` comprises thirty two four-core nodes. Each node includes two AMD 2214 processors, 8GB memory with 80GB local scratch disk space, while Gigabit Ethernet (1GbE) provides the network interconnections. This cluster is controlled by Linux CentOS 5.2, Rocks 5.1, and Portable Batch System (PBS) Torque 2.3.6. Users have unprivileged access to the machine, so they need to install any needed software (libraries etc) in user space. As the "home" environment for LifeV developers, this cluster was pre-provisioned with the entire set of packages required to run LifeV-based CFD simulations. Being an internal resource, with restricted user access, `puma` does not implement a monetary accounting system for computing resource usage.

[1] This is the "home" environment where the application is run by default.

B. Ellipse

The university cluster `ellipse` consists of 256 four-core nodes with AMD 2218 processors and 8GB RAM; Gigabit Ethernet provides the interconnection fabric. All nodes are controlled by CentOS 4.5. Job execution is performed by the Sun Grid Engine (SGE) 6.1 scheduler which was configured to manage serial processing batches only. As with `puma` users, `ellipse` users have unprivileged access to the machine. The required software dependencies were not originally installed on the cluster. They were provisioned by building them from sources in user space. All users pay a flat rate 5¢ per CPU core per hour.

C. Lagrange

Our third test architecture was the supercomputer cluster `lagrange` at the CILEA supercomputer center. This supercomputer, when assembled, was placed at the 136-th position in the TOP500 list [30]. The machine is composed of HP ProLiant server blades with two Intel Xeon X5660 processors and 24 GB RAM each. The network infrastructure is provided by InfiniBand (IB) 4X Double Data Rate (DDR, 20 Gb/s bandwidth). Computing nodes are controlled by the CentOS version 5.6 operating system. Users have unprivileged access to the machine. However, unlike `puma` and `ellipse`, `lagrange` provides some dependencies for LifeV-based applications (in particular the vendor-specific BLAS/LAPACK package). The cluster runs PBS Professional version 11 as a scheduler. The cost of the computer is €0.15 per core per hour (currently, about $0.20).

D. EC2

Our final target architecture was a infrastructure as a service (IaaS) cloud offered by Amazon EC2. IaaS resources provide on-demand computing in the form of computing chunks virtualized from the vendor's multi-tenant machines. These chunks are delivered for users as standard ssh-able root-accessible computational hosts. Users requesting these chunks specify the quantity of hosts, a resource class (characterized by computational power, number of CPU cores, memory capacity, and network interconnect) and the Operating System (OS) controlling the hosts (from *public* or users' *private* OS images). The vendor offers several sizes of virtualized hosts, ranging from small instances (`t1.micro`/`m1.small`; one 32bit CPU, below 1GB of RAM, and slow network interconnections) to modern HPC-class cluster nodes (`cc2.8xlarge`/`cg1.4xlarge`; 16 cores, 60GB of RAM, 2 GPGPU processors, and 10 Gigabit Ethernet (10GbE), with network-aware host allocation strategy [*placement groups*]). All setup conditions, as well as management and monitoring measures can be controlled by users in various ways, including direct interactions with the AWS (Amazon Web Services) Management Console web toolkit or Amazon EC2 API command-line tools [31], programming libraries [32], or frameworks providing higher level services over IaaS clouds [33], [34]. In contrast to conventional computational resources, EC2 users obtain full access to hosts instantiated on the Amazon's service. As a result, we could use a system package management tool (`yum`, in our case) and modify the system configuration. Amazon does not levy any upfront costs and charges users merely for the actual use of resources (time and computational power), external data transfers, and scratch space (size); however, some OS images and additional services (e.g., static IPs) incur additional costs. In this study, we focused on evaluating the `cc2.8xlarge` instance.

	`puma`	`ellipse`	`lagrange`	`ec2`
cpu arch. # cpu/cores	Opteron 2/2	Opteron 2/2	Xeon 2/6	Xeon 2/8
RAM/core	1GB	1GB	1.3GB	3.8GB
network	1GbE	1GbE	IB 4X DDR	10GbE
storage	OK	insufficient disk quota	OK	insufficient image mod
access	user space	user space	user space	root
support	full	very limited	limited	none
build env.	yes	yes	yes	none yum
compiler	GCC 4.3.4	GCC 4.1.2	GCC 4.1.2	none yum
dependencies	all	none src. install.	blas, lapack src. install.	none src. install.
MPI	Open MPI	none src. install.	Open MPI	none yum
parallel jobs	yes	no	yes	no
execution	PBS	SGE	PBS	shell

Table I
SPECIFICATION OF THE TEST ARCHITECTURES DIFFERENCES. IN COLOR: HOW WE ADDRESSED THE MISSING CAPABILITIES

VI. PORTING EXPERIENCES

Execution of our two applications on the target architectures requires (1) providing all software dependencies, (2) running the actual build program (make) that links against the appropriate libraries and produces the final executable file, and (3) providing the parallel execution environment.

A goal of this exercise to keep the porting effort to the absolute minimum possible. Thus, no changes were made to the application source codes. We utilized all compatible software that was already available on the target (even if it was not the latest version) and resorted to installation (preferably from package repositories) only if the dependency was missing or incompatible. In the `ec2` case, we had to commit a minimal configuration allowing aggregation of computational hosts for a single parallel execution.

Table I shows, in brief, the state of capabilities provided by the test resources before porting. Below, we provide a full report describing all the activities required to elevate the resource capabilities to the LifeV build and execution environment.

A. Puma

This computer is fully sustain the build and execution of LifeV-based applications. As the result, we needed to use a generic Makefile to create the executable. To launch the simulations, we used the PBS job submission command.

B. Ellipse

The Ellipse environment already provided the GNU compiler collection in a compatible version (4.1.2) with C, C++, and Fortran compilers, as well as all needed deployment toolkits. We began assembling dependencies by provisioning the MPI package (Open MPI 1.4.4). Then, MPI tools were used to build ParMETIS 3.1.1, HDF5 1.8.7, Trilinos 10.6.4, and SuiteSparse 3.6.1. Additionally, we provisioned the Boost libraries 1.47. For the BLAS/LAPACK package we resorted to CPU vendor-specific implementation, available as ACML [35] 4.0.1. The last step was updating the `Makefile` for the simulation applications and building them. All software preconditioning actions took about 8 man-hours of work by an experienced member of the LifeV developers team.

The SGE on `ellipse` was not configured to support parallel tasks; however, Open MPI could detect and liaise with SGE to start and end tasks on assigned nodes. Thus we were able to use SGE commands to reserve and submit `mpiexec` jobs.

C. Lagrange

CILEA presented a pre-prepared environment for building and executing parallel, MPI-based applications. The administrators provided a choice of C++ and Fortran compilers (GCC version 4.1.2 and Intel Compiler suites 12.1); MPI packages (Open MPI, Intel MPI), and BLAS/LAPACK routines were available from the CPU vendor-specific libraries (MKL [36]). In order to provision the software dependencies for our software, we used GCC to build the Boost libraries 1.47 and SuiteSparse 3.6.1. The remaining software dependencies (HDF5 1.8.7, ParMETIS 3.1.1, Trilinos 10.6.4, LifeV 2.0.0) were built against Intel MPI compiler wrappers. All the preparatory actions took about 8 man-hours for the LifeV developer.

D. EC2

To exercise the port of our software to EC2, we initially selected the `cc1.4xlarge` instance (when we started our experiments `cc2.8xlarge` was not available) and the *EC2 CentOS 5.4 HVM AMI* (`ami-7ea24a17`) image. To facilitate software preconditioning steps we used the `root` access. As this version of CentOS Linux contained obsolete versions of software, we began with an update of the system using the `yum update` command. The chosen image contains only the essential packages, with neither development software nor scientific library support. In order to provide the source code build environments, we installed, using `yum`, GCC 4.4.5, GFortran 4.4.4, libtool 1.5.22 (with autoconf 2.59, automake 1.9.6), and Open MPI 1.4.4. To install CMake 2.8 we resorted to a source code installation as the required version was not available from the repositories. After this phase, we downloaded all required dependencies as the source codes, built, and installed them: GotoBLAS2 1.13, LAPACK 3.3.1, Boost 1.47, HDF5 1.8.7, ParMETIS 3.2.0, SuiteSparse 3.6.1, Trilinos 10.6.4, and LifeV 2.0.0. After these preconditioning steps, building the simulation application was straightforward.

We also encountered cloud-specific issues not seen on traditional resources. One concerned ssh host mutual authentication to enable automatic launch of remote MPI processes by `mpiexec` requiring pre-generation and storage of keys. The second issue was related to configuration of the EC2 service. We modified the *security group* by enabling all intranet TCP ports to allow MPI processes intercommunication. Additionally, we required more disc space for staging the problem meshes (originally, the utilized image provided 20GB partitions). We could fulfill this requirement by instantiating the NFS service or using the *Elastic Block Store volumes* with copies of the files (one volume may be mounted to a single EC2 instance only). However, we decided to increase the size of the original boot partition, consequently supplying the input files from the same volume.

All the changes committed on the running instance can be preserved by creating a private image stored on the Amazon service. This image, in turn, may be used to launch several identical copies of the instance. Such on-demand hosts behave like cluster nodes. Further conditioning may provide a high-availability computing cluster with services such as monitoring or automatic checkpointing. However, we prepared an image that contains merely the essential software packages and services that allow the on-demand resource to sustain our CFD simulations.

In order to execute a simulation, we instantiated an appropriate number of copies of the prepared image. The service assigned intranet IP addresses for the on-demand hosts and we used these IPs to create the run-specific hosts list for the `mpiexec` command. Finally, this command was executed directly from the command line.

VII. Experimental Results

As mentioned, we benchmarked the four described architectures using two test cases: a simple RD test with boundary conditions specifying the exact solution on the boundaries; and a solution of the Navier-Stokes problem where, again, we prescribe the exact solution on the boundary.

A. RD test

We executed the simple RD 3-D problem on four computing architectures: two in-home clusters (`puma`, `ellipse`), the HPC-class computer (`lagrange`), and the on-demand

instantiated Amazon's hosts (ec2). In the case of EC2 resources, we utilized the newly introduced, most powerful Amazon's instance driven by two eight-core Intel Xeon E5 processors with 60.5GB of RAM (cc2.8xlarge). Though, this instance type was different from the build target, the transition was streamlined – the preconditioned image was fully compatible with both types and the compilers used generated optimized, binary-compatible executables. As this node type includes sixteen computing cores, it allowed us to conduct our experiment with a 200^3 element input mesh on 10^3 MPI processes on just 63 instances.

During the execution phase on ellipse and lagrange, we encountered system difficulties that limited our experiments on these targets. The former machine was not natively configured to execute the parallel jobs and our tasks spanning above 512 processes could not be launched (mpiexec was unable to initialize a huge number of remote MPI daemons). On the former target, our simulation codes reached the configured limit of data volume sent by the IB network adapters. As a result, we could not execute tasks bigger than 343 processes there.

In figure 4, we present results from a weak scaling test of the RD application. We started from a single process loaded with the input mesh of size 20^3 elements and incremented the number of processes (as well as the input mesh size) as cubic powers, to the limits of the platform in question. We recorded iteration wall-clock times across the whole MPI execution: the average times of assembly, preconditioning, and solver phases with the total maximal iteration time. We discarded timings from the first 5 iterations to guarantee that the acquired results are not influenced by Open MPI startup artifacts. Finally, all the consecutive measurements were averaged and are presented in the chart.

As shown in the figure, the problem scales well for all targets in the range 1-125 MPI processes – when the problem size is increased with the machine size, execution times remain reasonably steady (perfect weak scaling would result in constant times). We assume that network performance is the major factor degrading performance in the larger cases – as the problem size grows, processes exchange more data and the overall performance drops. After a certain problem size, only the HPC machine lagrange maintains a good weak scaling characteristic. However, we need to investigate why the solver phase on lagrange performs better for an increasing number of processes; we think that the placement of nodes in the cluster may play an important role in this phenomenon. Another interesting observation is that in case of Amazon's hosts, there are certain sizes where the performance significantly deteriorates. As different computation phases exchange different volumes of data (the assembly phase needs more data than preconditioning which needs more data that the solver) these characteristic locations appear for various numbers of processes. One more striking aspect is that the ec2 configuration characterizes by the worse performance degradation in comparison to puma and ellipse (both with 1GbE network). Due to the fact that each utilized EC2 instance incorporates sixteen CPU cores, the on-demand assembly exploits notably fewer hosts hence the smaller volume of data is exchanged by the 10GbE network.

$\#$ mpi	$\#$	$full$		mix	
		$time[s]$	$real\ cost[\$]$	$time[s]$	$est.\ cost[\$]$
1	1	4.83	0.0032	4.77	0.0007
8	1	5.83	0.0039	5.78	0.0009
27	2	7.28	0.0097	7.58	0.0023
64	4	8.69	0.0232	8.82	0.0053
125	8	21.65	0.1155	21.24	0.0255
216	14	31.47	0.2937	31.47	0.0661
343	22	66.34	0.9729	62.57	0.2065
512	32	92.20	1.9670	94.52	0.4537
729	46	127.76	3.9179	128.10	0.8839
1,000	63	162.09	6.8077	148.98	1.4079

Table II
COMPARISON OF TWO EC2 cc2.8xlarge ASSEMBLIES: FULL PAID INSTANCES IN A SINGLE PLACEMENT GROUP (FULL) AND SPOT REQUESTS IN VARIOUS PLACEMENT GROUPS (MIX)

B. Placement group benchmark

We also analyzed how the *placement group* setting influenced the performance of on-demand machines. In order to test this, we executed the RD code in two configurations, both utilizing the same cc2.8xlarge instances and preconditioned image. The first configuration exploited the fully paid 63-node assembly located in a single placement group, while the second configuration used 63 nodes acquired both from *spot requests* (instances sold for bid prices) and fully paid requests from four different placement groups in the same *availability zone* us-east-1a.

Table II presents the test results: the average total time for a single iteration and its cost in both configurations (during the test, the regular instance cost $2.40 and the spot-requested – 54¢, both prices per host per hour). The results show that regular allocation in a single placement group does not introduce any performance benefits despite costing four times as much. Of course, the unpredictable nature of spot requests makes it impossible to estimate when instances start, how long they are available, and their actual price (although a maximum can be specified). Indeed, we never succeeded in establishing a full 63-host configuration of spot request instances.

C. Navier-Stokes test

In figure 5, we present the weak scaling results achieved on our four basic test architectures, using the second application – Navier-Stokes 3-D simulation. We loaded computers as in the first case – every MPI process held 20^3 elements of the input mesh. As with RD, we could not execute this test on all available cores on ellipse and lagrange. As before, we also discarded the first few iterations to insulate

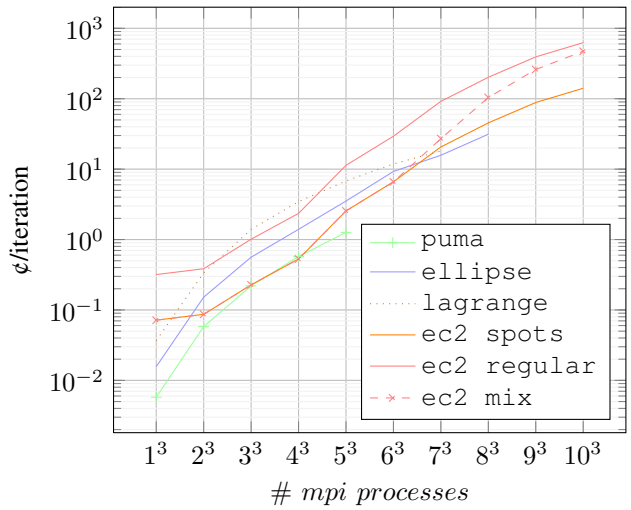

Figure 6. Per iteration costs of the test architectures for the RD application weak scaling benchmark

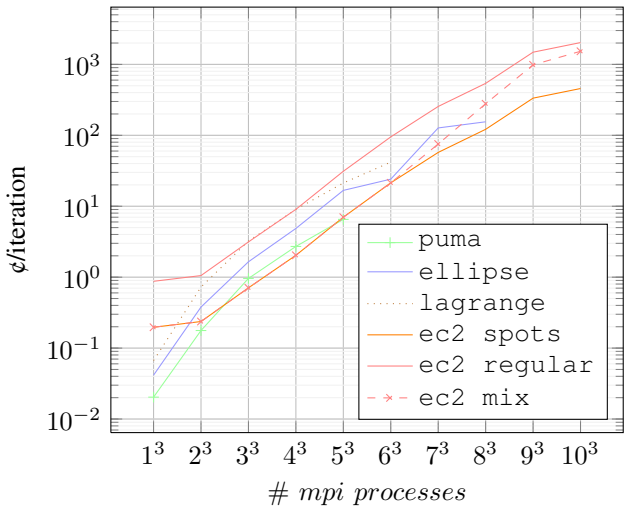

Figure 7. Per iteration costs of the test architectures for the Navier-Stokes application weak scaling benchmark

the timings from MPI startup impact; the chart presents averaged times for all observed iterations.

The Navier-Stokes test is more computationally demanding than the simple RD test. Moreover, the data volume exchanged among the MPI processes during the computation increases as this problem involves two variables. This test does not scale well in any range; however, again the most efficient machine is the HPC `lagrange` cluster. We believe that the results manifest the obvious explanation, i.e. that this type of CFD simulation is critically dependent on network performance. Again, the performance of Amazon cluster nodes declines sharply as the problem size/number of processes increases. However, for computationally intensive tasks for a small number of processes, Amazon EC2 performance is comparable to the HPC class machine and can considerably improve time to completion in comparison to the department class computing clusters.

D. Cost analysis

Figures 6 and 7 compare costs for resource utilization. We estimate the cost of our department cluster `puma`, based on its real capital cost and operating expenses, at 2.3¢ per core-hour, which is consistent with other published estimations [37]. For our university cluster `ellipse`, users pay a flat rate of 5¢ per core-hour. The cost per core for the 16-core EC2 instances applied in the study starts from 3.375¢, if spot requests are used, or 15¢ for flat-price nodes. However, as Amazon charges the users for the entire machine, this price increases if not all cores are utilized, as shown on both charts for two first cases. Finally, the cost of `lagrange` was set at 19.19¢ per core-hour based on the prevailing currency exchange rate.

Perhaps unsurprisingly, compute-intensive applications are most cost effective – one obvious reason being that neither clouds nor grids charge for network utilization and all charges are based on nodes. This is readily apparent in the case of the Navier-Stokes application – EC2 costs less than our on-premise cluster and is faster as well. Both figures contain the "`ec2 mix`" curves which could be viewed as a *cost-aware strategy* for Amazon's resources. However, obtaining a large number of hosts via spot requests is difficult if not impossible. In our experiments, we were compelled to add regularly-priced hosts to spot-request hosts to obtain the size configuration needed; this is apparent in the convergence of the mix and regular curves.

VIII. Summary and Future Work

We have presented preliminary experiences and observations based on our exercise to deploy two production CFD codes on four different target platforms characterized by heterogeneity in secondary attributes. Noteworthy are the difficulties that we had to overcome to simply provision the application, including package and library installation and other logistical hurdles.

Comparing on-premise and on-demand targets for the applications we tested, we found some evidence to support the claim that IaaS resources may be utilized for scientific CFD simulations possibly at lower cost than incurred locally. In particular, the spot-request feature coupled with availability of cutting edge resources (16-core nodes, 60GB RAM as opposed to 3-year old, 2-4 core nodes with 4GB RAM), suggests that small on-demand assemblies may be a viable alternative to local clusters. It is not without significance that IaaS's provide resources immediately, while local and grid resources are often subject to long queue wait times – an aspect that might offset any additional expense. Another factor is size; at least in our case, only Cloud providers could

provide a large enough offering to sustain the biggest, 1000-core task. Furthermore, while a modern local computing cluster, with an efficient interconnection network will outperform an on-demand assembly (which is highly vulnerable to network performance), the cloud solution might be useful for other reasons.

Another issue concerns the effort required for preparing the target platforms. In this study, we provisioned all machines manually and we installed only the necessary and sufficient packages. We observe that provisioning of a machine took about a day for an experienced member of the development team, in addition to multiple requests to and interactions with system administrators. Use of third party software to address mundane, repeatable tasks (e.g. [38]) or predefined images for IaaS ([33], [14]) could significantly reduce this cost and will form the focus of our future work.

ACKNOWLEDGMENT

The authors would like to thank Paride Dagna and the HPC group at the Consorzio Interuniversitario Lombardo per L'Elaborazione Automatica (CILEA). Their assistance in building and executing the applications on the HPC cluster has been extremely valuable.

REFERENCES

[1] I. Foster, Y. Zhao, I. Raicu, and S. Lu, "Cloud computing and grid computing 360-degree compared," in *Grid Computing Environments Workshop, 2008. GCE'08*, pp. 1–10, Ieee, 2008.

[2] "LifeV Project." http://www.lifev.org, 2012.

[3] S. Ostermann, A. Iosup, N. Yigitbasi, R. Prodan, T. Fahringer, and D. Epema, "A performance analysis of ec2 cloud computing services for scientific computing," *Cloud Computing*, pp. 115–131, 2010.

[4] C. Evangelinos and C. Hill, "Cloud computing for parallel scientific hpc applications: Feasibility of running coupled atmosphere-ocean climate models on amazons ec2.," *ratio*, vol. 2, no. 2.40, pp. 2–34, 2008.

[5] D. Gottfrid, "Self-service, prorated super computing fun!," *The New York Times*, vol. 1, 2007.

[6] S. Akioka and Y. Muraoka, "Hpc benchmarks on amazon ec2," in *Advanced Information Networking and Applications Workshops (WAINA), 2010 IEEE 24th International Conference on*, pp. 1029–1034, IEEE, 2010.

[7] E. Deelman, G. Singh, M. Livny, B. Berriman, and J. Good, "The cost of doing science on the cloud: the montage example," in *High Performance Computing, Networking, Storage and Analysis, 2008. SC 2008. International Conference for*, pp. 1–12, Ieee, 2008.

[8] "CAELinux." http://www.caelinux.com/CMS/, 2012.

[9] "Salome." http://www.salome-platform.org/, 2012.

[10] "Code_Aster." http://www.code-aster.org/, 2012.

[11] "Code_Saturne." http://research.edf.com/research-and-the-scientific-community/software/code-saturne/introduction-code-saturne-80058.html, 2012.

[12] "OpenFOAM." http://www.openfoam.com/, 2012.

[13] "Elmer." http://www.csc.fi/english/pages/elmer, 2012.

[14] "Using OpenFOAM with Amazon EC2." http://www.openfoam.com/resources/ec2.php, 2012.

[15] S. Salsa, *Partial differential equations in action: from modelling to theory*. Springer Verlag, 2008.

[16] L. Formaggia, F. Saleri, and A. Veneziani, *Solving numerical PDE's: problems, applications, exercises*. Springer Verlag, 2012.

[17] A. Ern and J. Guermond, *Theory and practice of finite elements*, vol. 159. Springer Verlag, 2004.

[18] S. Brenner and L. Scott, *The mathematical theory of finite element methods*, vol. 15. Springer Verlag, 2008.

[19] L. Quartapelle, *Numerical solution of the incompressible Navier-Stokes equations*, vol. 113. Birkhauser Basel, 1993.

[20] H. Elman, D. Silvester, and A. Wathen, *Finite elements and fast iterative solvers: with applications in incompressible fluid dynamics*. Oxford University Press, USA, 2005.

[21] C. Ethier and D. Steinman, "Exact fully 3d navier–stokes solutions for benchmarking," *International Journal for Numerical Methods in Fluids*, vol. 19, no. 5, pp. 369–375, 1994.

[22] S. Joachim, G. Hannes, and G. Robert, "NETGEN - automatic mesh generator." http://www.hpfem.jku.at/netgen, 2012.

[23] C. Geuzaine and J. Remacle, "Gmsh: a three-dimensional finite element mesh generator with built-in pre-and post-processing facilities," *International Journal for Numerical Methods in Engineering*, 2008.

[24] G. Karypis and V. Kumar, "HParaView." http://www.cs.umn.edu/~metis, 2012.

[25] Sandia National Laboratories, "The Trilinos Project." http://trilinos.sandia.gov, 2012.

[26] "HParaView." http://www.paraview.org, 2012.

[27] "SuiteSparse." http://www.cise.ufl.edu/research/sparse/SuiteSparse/, 2012.

[28] "Boost C++ Libraries." http://www.boost.org, 2012.

[29] The HDF Group, "Hierarchical data format version 5." http://www.hdfgroup.org/HDF5, 2012.

[30] "TOP500 Project." http://top500.org/, 2012.

[31] "Amazon Web Services/Developer Tools." http://aws.amazon.com/developertools, 2012.

[32] "boto - Python interface to Amazon Web Services." http://code.google.com/p/boto/, 2012.

[33] MIT, "StarCluster - Software Tools for Academics and Researchers." http://web.mit.edu/stardev/cluster/, 2012.

[34] J. Slawinski, M. Slawinska, and V. Sunderam, "The unibus approach to provisioning software applications on diverse computing resources," in *International Conference On High Performance Computing, 3rd International Workshop on Service Oriented Computing*, 2009.

[35] AMD, "AMD Core Math Library (ACML)." http://www.amd.com/acml, 2012.

[36] Intel, "Intel Math Kernel Library (Intel MKL)." http://software.intel.com/en-us/articles/intel-mkl, 2012.

[37] P. Smith, "A cost-benefit analysis of a campus computing grid," 2011.

[38] "DoIt Automation Tool." http://python-doit.sourceforge.net/, 2012.

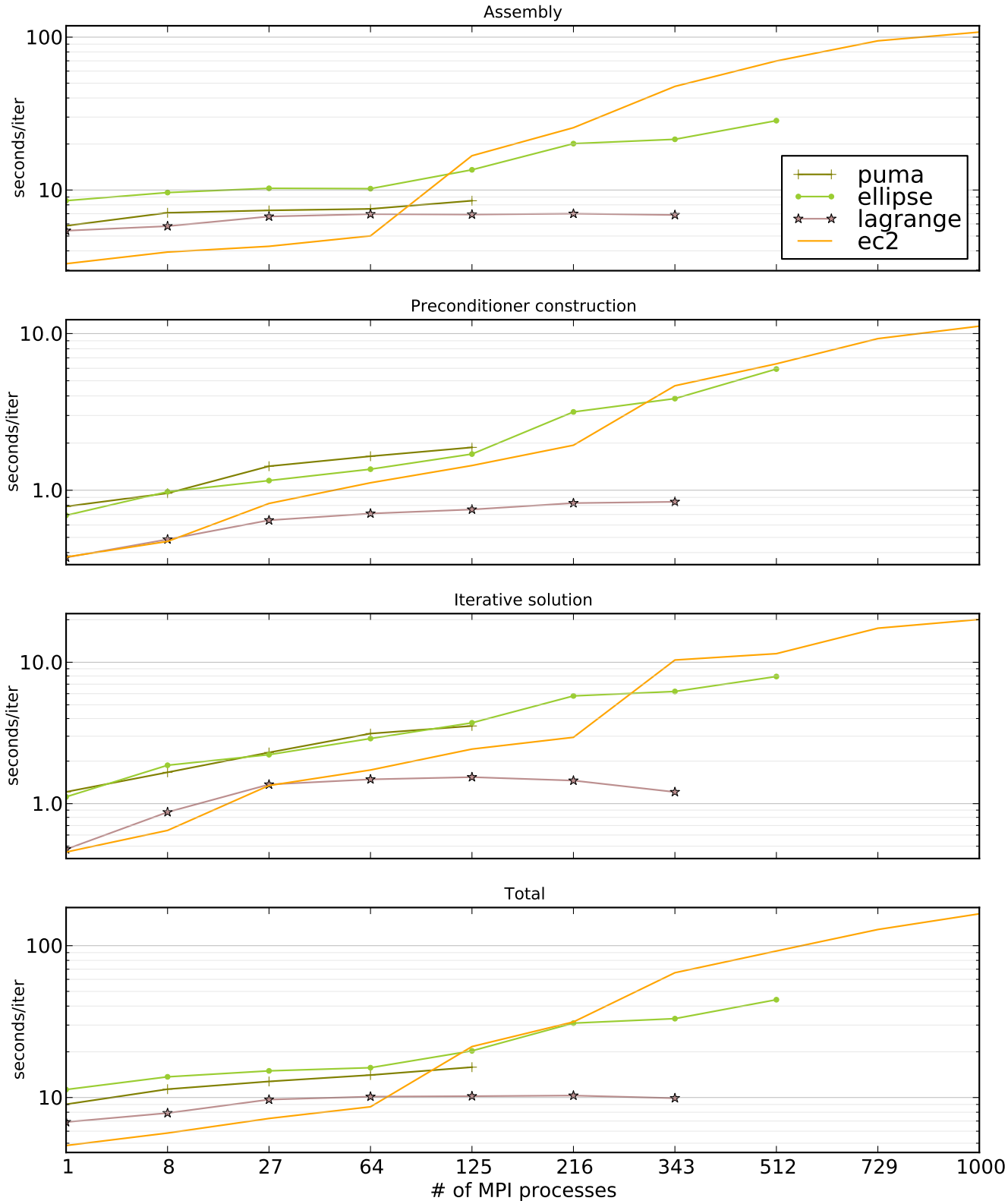

Figure 4. Weak scaling test of the RD 3-D simulation. The initial size of the problem mesh is 20^3

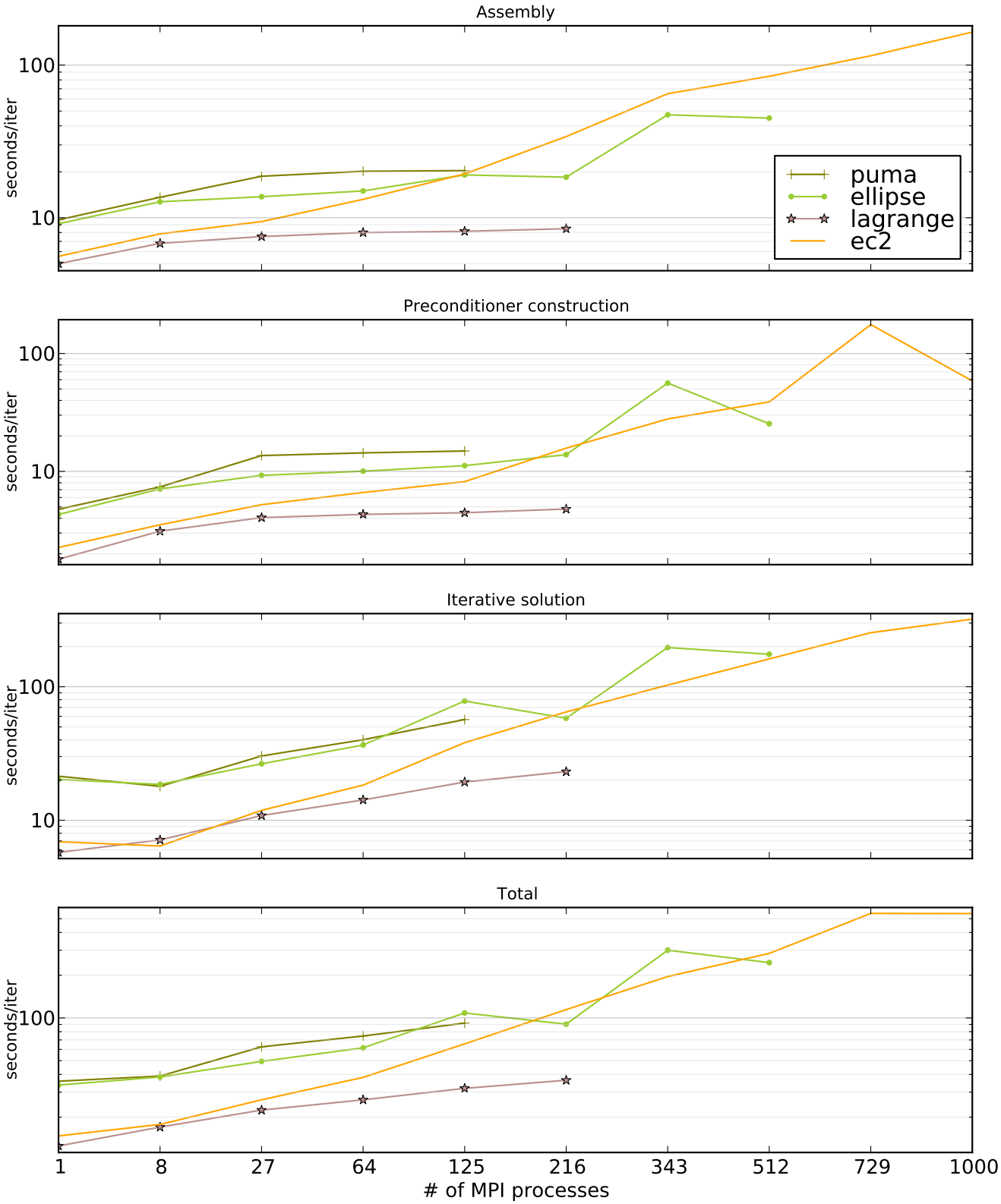

Figure 5. The weak scaling test of the Navier-Stokes 3-D simulation. The initial size of the problem mesh is 20^3