# An Adaptive Skeletal Task Farm for Grids

Horacio González-Vélez*

University of Edinburgh
School of Informatics
Edinburgh, UK
h.gv@ed.ac.uk

**Abstract.** Algorithmic skeletons abstract commonly used patterns of parallel computation, communication, and interaction. By demonstrating a predictable communication and computation structure, they provide a foundation for performance modelling and estimation. Grids pose a challenge to known distributed systems techniques as a result of their dynamism. One of the most prominent research areas concerns the availability of proved programming paradigms with special emphasis on the performance side. Thus, adaptable performance improvement techniques have been the subject of intense scrutiny. Scant research has been conducted on using the skeletal predicting information to enhance performance in heterogeneous environments. We propose the use of these predicting properties to adaptively enhance the performance of skeletons, in particular of a task farm, within a computational grid.
Hence, the problem addressed in this paper is: given a skeletal task farm, find an effective way to improve its performance on a heterogeneous distributed environment by incorporating information at compile time that helps it to adapt at execution time. This work provides a grid-enabled, adaptive task farm model, using the NWS statistical predictions on bandwidth, latency and processor availability. The central case study implements an ad-hoc task farm based on C/MPI and employs PACX-MPI for inter-node communication. We present initial promising results of parallel executions of an artificially-generated numerical code in a grid.

## 1 Introduction

With the advent of grid computing, the availability of proved programming paradigms has become an issue in computational science. It is widely acknowledged that one of the major challenges in programming support for these environments is the prediction and improvement of performance, due to the vast aggregation of heterogeneous resources and policies. Indeed, holistic projects emphasise the need not only for reliable programming environments, but also for improved performance capabilities [1, 2].

Performance enhancement is a multidimensional activity. One of the most powerful of these dimensions relates to optimisation techniques which work adaptively on an application-specific basis. Grid adaptability is quite broad, and often

* Work partly supported by the EC-funded project HPC-Europa, contract number 506079. Special thanks are owed to Murray Cole for his suggestions and review. The comments of anonymous referees have also helped to improve this paper considerably.

J.C. Cunha and P.D. Medeiros (Eds.): Euro-Par 2005, LNCS 3648, pp. 401–410, 2005.

relates to flexibility, ability to transform and evolve, re-usability, and extensibility. It encompasses several layers in the architecture, involves a multiplicity of parameterised values, and is typically performed in a custom-built fashion either at compile-time or during execution. There have been substantial examples of the applicability of adaptable models in computational grids [3, 4]. Moreover, AppLeS [5] builds on these efforts and provides a comprehensive approach including resource discovery, selection, and scheduling . Nevertheless, one of the most fascinating open-ended questions in computational science still concerns the self-adaptation of programming structures to grids [6].

The separation of software and hardware has long been considered critical to the success of any parallel programming endeavour, as it is vital to foster the reuse of algorithms and software. Furthermore, we consider that the division of the structure from the application itself to be crucial to the goal of delivering adaptability.

Algorithmic skeletons (AS) abstract commonly used patterns of parallel computation, communication, and interaction [7]. They present a top-down structured approach where parallel programs are formed from the parametrisation of skeleton nest, also known as Structured Parallelism (SP). AS provide a clear and consistent structure across platforms by distinctly decoupling the application from the structure in a parallel program [8]. They do not rely on any specific hardware and benefit entirely from any performance improvements in the systems infrastructure.

SP is not universally applicable to the production of parallel and distributed programs, but there is an important growing number of applications to consider them an interesting research area [9]. Furthermore, skeletal methodologies inherently posses a predictable communication and computation structure, since they capture the structure of the program. They provide, by construction, a foundation for performance modelling and estimation of parallel applications.

This work is concerned with the feasibility of using this predicting capabilities of SP. In particular we propose a model to enhance the performance of skeletal task farms in grids. First, we consider related work and provide motivation, moving on to some scalability results using the farm of the Cole's eSkel [10] library. These figures are collected using a computational grid with nodes in Edinburgh and Stuttgart. This part helps us to reinforce the feasibility of the skeletal programming model for grids. Then, we have constructed our main case study using an utilitarian skeletal task farm implemented using C, MPI and the PACX-MPI [11] library for the inter-cluster communication. It also employs the Network Weather Service (NWS) [12] to monitor the grid environment, forecast processor and network availability, and adapt to the load conditions on the nodes and interconnections. While the model is embedded in the code, the NWS forecasts drive the behaviour adaptation of the task farm at execution time. Finally, future directions for this work are supplied.

## 2 Motivation

We would like to formulate the problem covered in this paper as: given a skeletal task farm, find an effective way to improve its performance on a heterogeneous

distributed environment by incorporating information at compile time that helps it to adapt at execution time.

Compile-time optimisers formulate static decisions about the expected behaviour of an application. On the other hand, run-time optimisers do not generally possess direct knowledge of the structure (meaning) of the application. They lack specific information on its data and control flows and their operation is normally driven by load-balancing criteria. Indeed, the development of effective compilers and optimisers remains an active area in computer science.

In addition to this, there is no equivalent to a compiler or to an universal run-time optimiser for grids. Due to the complexity involved, grid optimisation techniques have usually been custom-made. They require the modification of the source code to enable its operation [5], the use of capacity characterisation methods [13], or even the creation of special-purpose languages [14] .

From a grid perspective, although different parallel solutions have traditionally exhibited skeletal constructs, their associated optimisations have not employed the structural information of the skeletons but rather modified the scheduler [15], or have not decoupled entirely the structure from the behaviour keeping the actual application interlaced [16]. On the AS side, the emerging approaches to performance optimisation in computational grids have employed process algebra methods [17] and future-based RMI mechanisms with Java [18].

AS possess a crucial property which favours performance optimisation: their structured and predictable behaviour for a given meaning (program). Nevertheless, scant research has been conducted on improving the skeletal performance by actively using this information from a systems infrastructure perspective.

Thus we would like to bring all these factors together and use the structural, forecasting capabilities of skeletons to optimise pragmatically their performance in grids. As opposed to standard optimisation at compile time, in this case the behaviour of the application is known prior to the execution. In contrast to standard run-time optimisation, the meaning is clearly defined as well. In principle, therefore, this skeletal optimiser could forecast and enhance the actual behaviour of the application by exploiting the knowledge of its structure.

The open question must be: how much can the structural forecasting information of AS help to improve the performance of parallel applications whilst executing in heterogeneous clusters and eventually in computational grids?

Hence, we argue that the AS need to evolve to include adaptive capabilities to improve its performance in the Grid. In this work, we present a pragmatic approach using an optimised ad hoc task farm skeleton, NWS, and a realistic grid environment. It is important to note that we do not intend to develop a scheduler nor to solve the general optimisation case for every structured parallel program. We concentrate on empirically finding optimisation techniques for solutions based on skeletal task farms.

This work is the first step in the development of a framework which incorporates different SP programs and its associated optimisation techniques, similar to the way a compiler incorporates optimisation techniques, to be deployed in computational grids as illustrated in Fig. 1. The ultimate objective of this ongo-

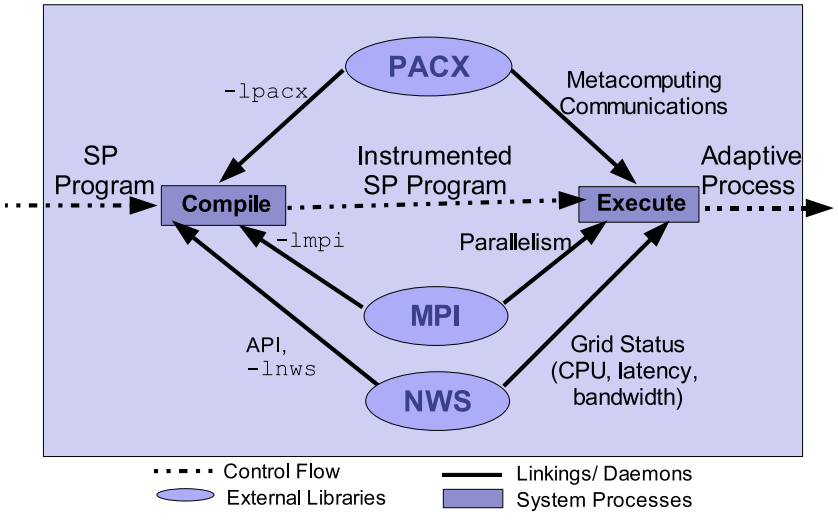


**Fig. 1.** Methodology to Introduce Adaptiveness into Structured Parallelism Programs

ing endeavour is to build upon theoretical performance models and design a set of adaptive techniques. The main difference to other performance approaches is that it intends to be SP-oriented, adaptable by construct, and focused on empirical, system-infrastructure methodologies.

## 3 The Task Farm

We have selected a task farm for this initial approach due to its applicability to the solution of a great number of problems in parallel programming and its simple structure.

A Task Farm (TF) can be roughly described as a "farmer" process which spawns $N$ independent "worker" processes to execute a parallel workload. The TF is composed by a input $I$, an output $O$ and a function $F$, or $TF = \langle I, O, F \rangle$. A worker executes a task by mapping $F$ into a subset of $I$ (task size), computing a subset of $O$, and then reporting back to the farmer for the next unit of work or termination. This is shown schematically in Fig. 2.

The TF construct has traditionally dealt with fine-grained data parallelism. In its canonical form, the communication times between the farmer and the workers can be adjusted to be constant and much less than the computation times [19]. All workers are allocated to dedicated processor in a parallel machine and the computation of each element $O$ is independent and characterised by the fact that the $F$ does not generate the same amount of work for different elements. The TF needs to keep distributing the elements in $I$ to avoid worker starvation while minimising communication. This TF feature aims at providing the best load-balancing. Under this scenario, the number of elements of $I$ sent to a given worker at once (task size) can be statically calculated to minimise idle times and require minimal scheduling from the farmer side [19].

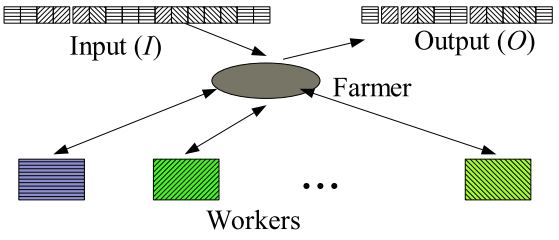


**Fig. 2.** A Task Farm

However, in more realistic scenarios, the farmer requires to assign different tasks sizes to workers because:

- The underlying architecture can have multiple communication links between the farmer and workers with different bandwidths and latencies
- The workers and the farmer run in non-dedicated nodes with distinct workloads in a distributed environment

Furthermore, in a computational grid, communication and computation times vary greatly, an undersized farmer, a saturated communication channel, or the sudden termination of a worker can produce unpredictable situations. Thus our particular objective is to adaptively determine the optimal task size for each worker in a task farm in order to minimise the total execution time for a given application.

## 4 Implementation

We have initially employed the first version of the Cole's eSkel library and an artificial integer application. The overall system is configured by evenly distributing the processes between two 16-node Beowulf clusters located at the High Performance Computing Centre (HLRS) in the University of Stuttgart and the School of Informatics in the University of Edinburgh respectively.

The farmer node has been positioned at HLRS. We have used the PACX-MPI library for interconnection with the allocation of two pairs of communication nodes to interconnect both installations. This is the standard requirement for soundly executing PACX-MPI. The MPI versions are MPICH and LAM/MPI and the nodes and communication channels are in non-dedicated mode.

Figure 3 shows the channel utilisation from the worker standpoint on the eSkel Task Farm version 1.0 and PACX-MPI 5 for a simplistic application. It presents different values of $I$ ranging from 160B or $I = 200$ to 1.6MB or $I = 200,000$. It is important to mention that half of the workers are located in Edinburgh while the other are in Stuttgart.

Although the communication channel at 270KB/s is not saturated while working with 8 processes and 1.6 MB and 160KB vectors, equivalent to an $I$

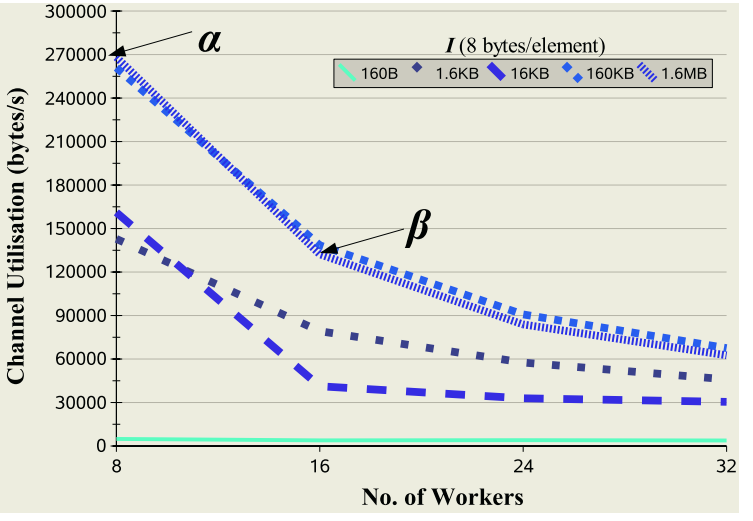


**Fig. 3.** Worker Channel Utilisation for different values of $I$, ranging from 160B or $I = 200$ to 1.6MB or $I = 200,000$, using the eSkel Task Farm version 1.0 and PACX-MPI 5. Half of the workers are located in Edinburgh and the other half in Stuttgart

of 200,000 and 20,000 8-byte data elements respectively ($\alpha$ in Fig. 3), the increase to 16 processes with the same amount of data implies a 50% decrement in the ability to transmit ($\beta$ in Fig. 3). This is unexpected since there are more processes transmitting than the decrease in channel utilisation.

After the analysis of the communication patterns when increasing the number of processes, the performance degradation in the communications was attributed to the use of MPI collective communication operations under PACX-MPI, since the synchronisation mechanisms in PACX-MPI are not optimised for collectives across sites.

In order to address this issue, we have implemented a new skeletal TF using MPI send-receive operations only. Furthermore, this new TF incorporates the ability to adapt the task size using the NWS forecasting capabilities of the task farm.

NWS provides utilitarian forecasting figures based on the statistical time-series analysis of the processing and networking load and configuration. It presents fault-resilient capabilities to support adaptive applications, and its accuracy has been successfully tested in major grids [20].

This case study features two distinct approaches to define the task size: static and adaptive. In the former, the task size assigned to a worker is defined by the full input data size and the number of workers. Although each worker operates on the same task size, except possibly for the last one, even processing is not guaranteed due to the different node workload and network conditions.

In the adaptive strategy, the central part of this work, the task size is defined by taking into account four factors:

- Available CPU: CPU fraction allocatable to a new process
- Bandwidth: Speed to transmit data to/from the farmer and a worker
- Current CPU: CPU fraction usable by a running process
- Latency: Time (in msec) to send a TCP message from the farmer to a worker

The forecasts of the above system indicators, supplied by NWS, are composed into a fitness index $FI$ as shown in (1). $FI$ defines the adaptive task size assuming a certain system behaviour based on historic measurements. It is also important to mention that the heuristic to calculate the index coefficients is application dependant, and in this initial approach, based on an artificially-created application, we have employed $A = 0.4$, $B = 0.1$, $C = 0.4$, and $L = 0.1$. The intention is to have an even task processing by allocating larger tasks to the fitter nodes in terms of its processing and communications.

$$FI_i = A * avail_i + \frac{B * bandwidth_i}{max(bandwidth)} + C * current_i + \frac{L * min(latency)}{latency_i} \quad (1)$$

The algorithm provides default values which assume an unfit node, since unexpected surges in workload and latencies may affect the operation of the NWS sensor and memory processes returning no readings at execution.

## 5 Empirical Results

We have deployed the initial implementation employing a configuration including 32 processors distributed into two 16-node Beowulf clusters (`bw240` and `bw530`) located at the School of Informatics in the University of Edinburgh. A summary of the hardware and software configuration of a typical node in each cluster is presented in Table 1.

**Table 1.** Hardware/Software Configuration of the `bw240` & `bw530` Beowulf Clusters located at the School of Informatics in the University of Edinburgh

| **CONFIGURATION** | `bw240` | `bw530` |
|---|---|---|
| CPU | Intel Pentium 4 1.80GHz | Intel Xeon 1.70GHz |
| Memory | 1 GB | 2 GB |
| Linux kernel | 2.4.20-24.7_1.public.1 | 2.4.20-31.9_v1_dice_1 |
| gcc | gcc-2.96-112.7.1 | gcc-3.2.2-5 |
| LAM/MPI | lam-6.5.6-tcp.1 | lam-6.5.8-4 |
| PACX | PACX-5.0-beta 9/8/04 | PACX-5.0-beta 9/8/04 |
| NWS | 2.10.1 | 2.10.1 |

All nodes were on non-dedicated mode during all the experiments, and their interconnection channels did not have any bandwidth reservation. We observed that the workloads, latency and bandwidth varied greatly during the experimentation periods.

There were NWS sensors running in all nodes, and there was a clique encompassing all nodes. The NWS name and memory server daemons were running in

the farmer node. We allocated four nodes (two per cluster) to run the PACX-MPI communication processes. All execution time measurements were obtained using the `MPI_Wtime` function for the TF skeleton only.

The farmer, the workers and the two-pair communications hosted concurrently the series of jobs for the static and adaptive runs for each $I$ (and therefore under similar external load and competing for the same resources). All farmer and worker had a system priority of 10.

Figure 4 graphs the execution times for $I = 512, 1024, 2048, 4096, 16384$. All entries in the graph perform $O(10^{12})$ daxpy operations, and present 8 different lines corresponding to $4, 8, 16$, and 32 workers for the static and dynamic models. The thicker lines average the series of executions for both models.

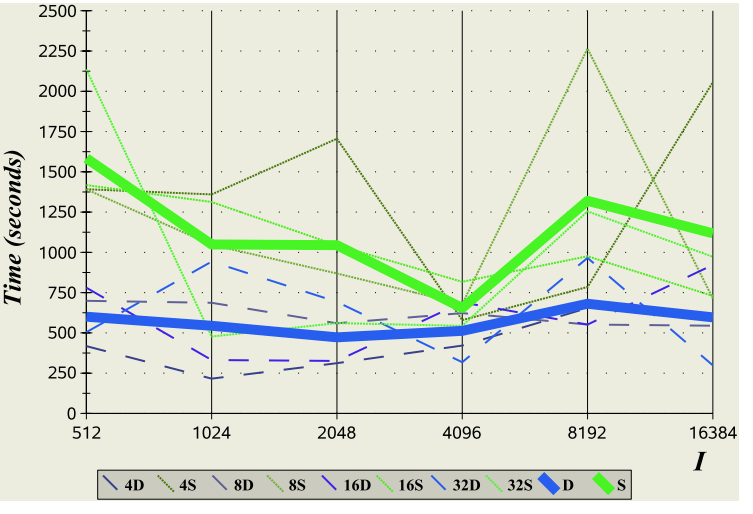


**Fig. 4.** Task Farm execution times (in seconds) for a series of concurrent executions. The application performs $O(10^{12})$ daxpy operations. Key: [numb][model], e.g., *4S* means 4 workers and Static model. The thick lines are the average of all executions

In an homogeneous dedicated system, one would expect a smooth line. That is to say, the peaks in the execution times in both models are chiefly defined by the non-deterministic nature of this grid, e.g. the noticeable peak in the static case with 8 workers and $I = 8192$. Hence, it is clear that the adaptive model surpasses the static one, as reflected by the fact that its averaged graph is flatter and with lower time measurements for every entry of $I$.

## 6 Future Directions

Although the nodes involved in this experiment demonstrated a certain degree of homogeneity within the same Beowulf cluster, their extremely different workload

and interconnections made them distinct enough to become a representative environment for this initial case study. Even more, the uncontrolled workloads present at the execution time comprised a non-deterministic scenario. On the other hand, the model still has room for improvement on its interaction with the NWS API.

The skeletal approach has helped us to discretise and bound the parameters involved, reducing the number of combinations. The use of AS allowed us to make assumptions on the input and output sets that permit a more effective allocation of computation and communication resources.

These results may seem to be intuitive. However, we consider them significant since they provide a common ground to further tweak our model under the dynamic environment of a computational grid. We intend to keep improving it by:

- Devising a more accurate adaptiveness strategy through more comprehensive experimentation. A biomedical code is being tinkered with [21].
- Deploying a faster distribution and execution of tasks by analysing the scalability, buffering, and resource monitoring issues.
- Improving the model by incorporating new indicators such as task termination time and CPU capacity. This in turn will provide foundations to develop a methodology for the creation of the index heuristics.

## References

1. Foster, I., Kesselman, C., eds.: The Grid: Blueprint for a new computing infrastructure. Second edn. Morgan Kaufmann, San Francisco, USA (2003)
2. Laforenza, D.: Grid programming: some indications where we are headed. Parallel Comput. **28** (2002) 1733–1752
3. Vetter, J.S., Reed, D.A.: Real-time performance monitoring, adaptive control, and interactive steering of computational grids. Int. J. High Perf. Comput. Appl. **14** (2000) 357–366
4. Cheng, S.W., Garlan, D., Schmerl, B., Steenkiste, P., Hu, N.: Software architecture-based adaptation for grid computing. In: HPDC-02: 11th IEEE Int Symp on High Performance Distributed Computing, Edinburgh, UK, IEEE CS (2002) 389–398
5. Berman, F., Wolski, R., Casanova, H., Cirne, W., Dail, H., Faerman, M., Figueira, S., Hayes, J., Obertelli, G., Schopf, J., Shao, G., Smallen, S., Spring, N., Su, A., Zagorodnov, D.: Adaptive computing on the grid using AppLeS. IEEE Trans. Parall. Distrib. Sys. **14** (2003) 369–382
6. Vadhiyar, S.S., Dongarra, J.J.: Self adaptivity in grid computing. Concurrency Computat. Pract. Exper. **17** (2005) 235–257
7. Cole, M.: Algorithmic Skeletons: Structured Management of Parallel Computation. MIT Press, London, UK (1989)
8. Cole, M.: Bringing skeletons out of the closet: a pragmatic manifesto for skeletal parallel programming. Parallel Comput. **30** (2004) 389–406
9. Rabhi, F.A., Gorlatch, S., eds.: Patterns and skeletons for parallel and distributed computing. Springer-Verlag, London, UK (2003)

10. Cole, M.: eSkel: The Edinburgh Skeleton library API reference manual. University of Edinburgh, UK. First edn. (2002)
Available on-line at: http://homepages.inf.ed.ac.uk/mic/eSkel.
11. Keller, R., Gabriel, E., Krammer, B., Muller, M.S., Resch, M.M.: Towards efficient execution of MPI applications on the Grid: Porting and optimization issues. J. Grid Comput. **1** (2003) 133–149
12. Wolski, R., Spring, N., Hayes, J.: The Network Weather Service: A distributed resource performance forecasting service for metacomputing. Future Gener. Comput. Syst. **15** (1999) 757–768
13. Hey, A.J.G., Papay, J., Surridge, M.: The role of performance engineering techniques in the context of the grid. Concurrency Computat. Pract. Exper. **17** (2005) 297–316
14. Lian, C.C., Tang, F., Isaac, P., Krishnan, A.: GEL: Grid execution language. J. Parallel Distrib. Comput. **65** (2005) 857–869
15. Casanova, H., Kim, M.H., Plank, J.S., Dongarra, J.J.: Adaptive scheduling for task farming with grid middleware. Int. J. High Perf. Comput. Appl. **13** (1999) 231–240
16. Shao, G., Berman, F., Wolski, R.: Master/slave computing on the grid. In Raghavendra, C., ed.: HCW'00: 9th Heterogeneous Computing Wksp, Cancun, Mexico, IEEE CS (2000) 3–16
17. Benoit, A., Cole, M., Gilmore, S., Hillston, J.: Scheduling skeleton-based grid applications using PEPA and NWS. Comput. J. **48** (2005) 369–378
18. Aldinucci, M., Danelutto, M., Dünnweber, J., Gorlatch, S.: Optimization techniques for skeletons on grid. In Grandinetti, L., ed.: Grid Computing and New Frontiers of High Performance Processing. Elsevier Science (2005) To appear.
19. Hey, A.J.G.: Experiments in MIMD parallelism. Future Gener. Comput. Syst. **6** (1990) 185–196
20. Wolski, R.: Experiences with predicting resource performance on-line in computational grid settings. Sigmetrics Perform. Eval. Rev. **30** (2003) 41–49
21. Gonzalez-Velez, V., Gonzalez-Velez, H.: A grid-based stochastic simulation of unitary and membrane $Ca^{2+}$ currents in spherical cells. In: 18th IEEE Int Symp on Computer-Based Medical Syst., Dublin, Ireland, IEEE CS (2005) To appear.