Managing the Construction and Use of Functional Performance Models in a Grid Environment

Robert Higgins, Alexey Lastovetsky
School of Computer Science
University College Dublin
Ireland
{robert.higgins, alexey.lastovetsky}@ucd.ie

Abstract— **This paper presents a tool, the Performance Model Manager, which addresses the complexity of the construction and management of a set of Functional Performance Models on a computing server in a Grid environment. The operation of the tool and the features it implements to achieve this goal are described. Integration of Functional Performance Models with a GridRPC middleware, using the tool's interfaces is illustrated. Finally, an example application is used to demonstrate the construction of the models and experiments that show the benefit of using the detailed models are presented.**

Keywords-grid computing; heterogeneous computing; performance modeling; benchmarking; processor performance; memory heirarchy;

I. INTRODUCTION

Grid Computing often utilizes highly heterogeneous networks of computers. Efficient high performance computing on Grids can only be achieved when accurate models of the performance of compute nodes are available to the scheduling middleware. The performance of a processor executing a problem is determined not only by the physical characteristics of the processor, but also of the nature of the problem's core algorithm and the size of the problem's input parameters. When scheduling the execution of a remote problem on a Grid, it is important to make an accurate estimation of the problem's execution time on the available heterogeneous processors.

In GridRPC [1] systems, many scheduling decisions are based on an estimation of execution time of the problem. The Minimum Completion Time heuristic [2] as implemented in NetSolve [3] or the Historical Trace Manager [4] heuristics implemented in GridSolve [5] both rely on the accuracy of this estimation. Presently, this is provided by the combination of a simple LINPACK [6] style benchmark, which measures operations per second of the processor, and a measure of the complexity of a given problem. This kind of estimation assumes that the core algorithm of the problem is similar to the benchmark code used, which is not necessarily the case. A problem may be more or less suited to a particular processor as a result of the underlying architecture of that processor. This difference may not be represented in the benchmark. Further, the single benchmark does not account for variance in processor speed as problem size increases. It is assumed that the speed of all processors will decrease at the same rate, which is never quite true, especially where paging occurs on processors at different sizes of problem.

The Functional Performance Model [7] (FPM) is a excellent candidate for making a more accurate estimation of a problem's execution time. It is a problem specific, realistic, experimentally obtained model of the actual execution speed of a problem expressed as a piece-wise linear function of the problem size. The problem size is given by its input parameters. These properties of the FPM address both issues with the current estimations used in GridSolve. However, the tasks of construction, management and use of a set of functional models are not trivial. This paper presents a tool that addresses these issues: the Performance Model Manager (PMM), and its integration with extended GridRPC system: SmartGridSolve [8].

In Section 2 we describe the FPM construction method, the installation of a problem into the manager, the management of model construction and interfaces to access constructed models. Section 3 describes the modification to GridSolve enabling use of constructed FPMs. Section 4 describes a GridRPC application that we use to demonstrate the construction and use of FPMS. Section 5 presents the constructed models and experimental results describing scheduling improvements as a result of using the tool.

II. PERFORMANCE MODEL MANAGER

PMM is a tool that has been developed to address issues surrounding the construction, maintenance and use of Functional Performance Models in a variety of parallel computing environments. It consists of three main features. Firstly it implements the Geometric Bisection Building Procedure [9] for multi-parameter FPMs, optimizing the construction of a problem's performance model. It permits a large number of problems to have their construction managed by implementing a flexible benchmarking scheduler, suitable for use where a queuing system does not exist. Finally it provides access to the models in a variety of ways, allowing feedback from actual executions and providing tools to use the models in scheduling decisions.

A. *Efficient Construction*

Un-optimized construction of the FPM for a large set of problems installed on a large number of servers is infeasible

978-1-4244-3750-4/09/$25.00 ©2009 IEEE

due to the time and resources that would be consumed. A novel algorithm that optimizes the construction has been described in [9], titled Geometric Bisection Building Procedure (GBBP).

Single parameter GBBP optimizations are made possible by using the natural variation in performance (due to a server's external load fluctuations) and assumptions on the shape of a FPM (that it may initially be increasing, but is then decreasing and monotonic). Examples of the shape of models that fit these assumptions are shown in Fig. 1.

1) Band of Performance

The performance of a server in a non-dedicated environment is variable and a performance model for such a server must not be static. A single FPM can be considered as a possible level of performance that a server may return under certain load conditions. When those load conditions are variable, the FPM becomes a band of performance levels rather than a single function.

GBBP finds a piecewise approximation of this band (illustrated in Fig. 1). The approximation is constructed in such a way that its intersection with the real-life performance band forms a simply connected surface. I.e. the approximation intersects the real-life band across the entire problem size range leaving no gaps. This ensures the accuracy of the model while allowing the formulation of optimizations that do not violate the constraint.

A history of load fluctuations, which are external to any Grid executions, is recorded. This history can be used to predict the maximal and minimal expected loads a problem of a particular size might encounter on execution. Benchmarks made during the construction of the model are adjusted by the loads they are predicted to encounter. The result is a maximal and minimal speed for every problem size and these form the band model.

The functional model is extracted from the midpoint between the limits of the band. The band itself is not used in scheduling as has been found to provide negligible benefit while adding a great deal of overhead. It's main purpose is in enabling the optimization of construction.

2) Model Shape

The optimization of construction is based on assumptions on the shape of the model. These are: that the performance may initially increase, then it will be decreasing and monotonic. The initial performance increase is discovered through a series of short benchmarks for small problem sizes. These are inexpensive.

Once non-increasing, the problem size range is recursively bisected. The performance at each bisection point is found through experimentation. At each point, an attempt is made to determine if further construction is required or if sufficient detail has been resolved in the model. Identifying where benchmarks are no longer needed minimizes the construction time.

For instance, in Fig. 2., when the model at benchmark points (a) and (b) is examined, we find that the vertical component of the band at (a) contains (b) entirely. As a result of the assumptions on shape (that it is monotonic), construction between these points can cease without violating the simply connected property of the real/model intersection.

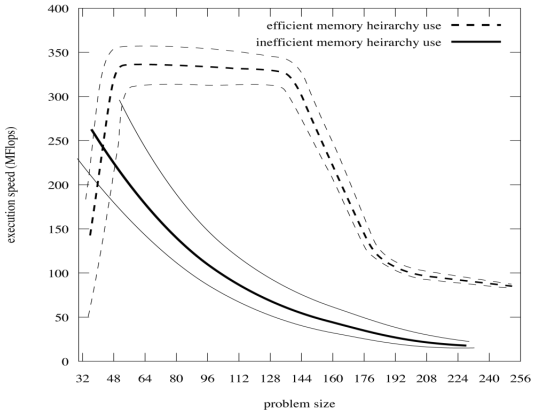

Figure 1. The typical profile of Functional Performance Model according to their memory access efficiency

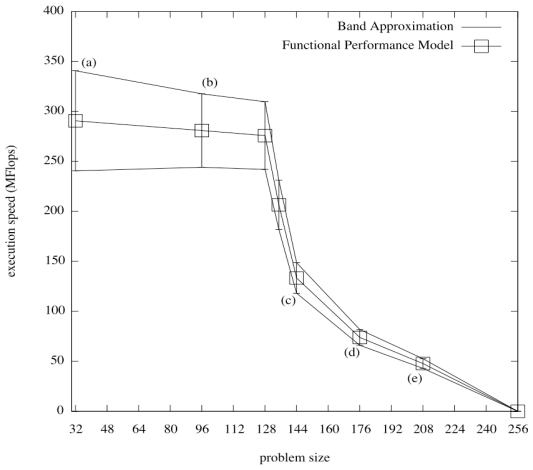

Figure 2. Functional Performance Models for the *barmatter* problem, with points for GBBP and a naïve construction method shown.

Further, in the case of benchmarks (c),(d) and (e) we can see that (e) was previously approximated by the segment joining (c) to (d) and, before that, by the segment joining (c) to the endpoint. Again, no further benchmarks are required in the intervals between these points, as it has been shown that the model adequately approximates the real band in these regions.

3) Multi Parameter

Multi parameter GBBP is a basic extension of the single parameter algorithm. A single parameter model is constructed for each problem argument with all other arguments fixed at their minimum sizes. These single parameter models form the boundaries of the full multi parameter model. Once their construction is complete, their points form a grid in the multi parameter model where experimentation must be made.

Arguments that do not affect the execution speed of a problem should not be included in the configuration of a problem in PMM, so as to avoid unnecessary experimentation. However, if they are included, the single parameter boundary model for these arguments will have a

unique flat profile that may be detected. Those boundary models with flat profiles can be excluded from the multi-dimensional grid of the full model.

4) PMM / Problem Interface

PMM provides the developer of a problem with a framework for using GBBP to construct the problem's FPM. An interface between a call to the problem and PMM must exist so that benchmarks can be executed automatically at points determined by GBBP.

To realize this interface, we have chosen to specify that the problem developer must provide a benchmarking binary that executes a call to the problem for us. This binary must follow a set of rules regarding the input that it accepts and the output it returns to PMM. In the specific environment of GridSolve, we could conceivably execute problems automatically without this requirement, as GridSolve already provides wrapper binaries for executing problems as GridRPC calls. However, there would be no facility to pass intelligible data for the problem to process. This would require a language additional to the Interface Description Language (IDL), which at present facilitates executing problems with data that has been passed to GridSolve. Such a language is likely to be complex to use and limiting for the problem developer. As a result, we believe the task of writing a small benchmarking binary is a far simpler and more flexible solution to the interfacing between the problem and PMM.

In order to allow the PMM to execute benchmarks of a problem at points as requested by GBBP, the benchmark binary of the problem that the developer provides must:

- accept an ordered list of command line parameters that define the size of the input parameters to the problem
- dynamically allocate input and output data structures according to the input arguments
- initialize input parameters with data that is intelligible to the problem call, and permits normal execution
- place calls to the PMM timer functions directly before and after execution of the problem
- terminate and return normally on successful execution

An example of a problem benchmark is shown in Fig. 3. The calls to PMM timing code are highlighted on lines 24 and 29. Also shown are constructors and destructors and the function that formats and prints the measured benchmarking information (line 31), which is parsed by PMM.

The problem that is benchmarked by this example is an N-Body simulation of dark matter. It comes from an application that will be described in section 4. The darkmatter problem acts on two large 3-dimensional matrices. Both these matrices must be cubic and as a result we only need to build the FPM in terms of two parameters, the size of a single side of each matrix, N_x and N_p. This is an important optimization that the problem developer can enable us to use, as greater numbers of parameters results in far longer construction time. There is also a constraint that N_p is less than N_x. The benchmarking binary can identify constrained parameters to PMM by returning a defined code as on line 14.

```
1  #include <pmm_util.h>
2  #include <"hydropad_bench.h">
3
4  int main(int argc, char **argv) {
5
6      /* declare variables */
7      global_data *gb;
8      int nx, np;
9      struct pmm_timer *t;
10
11     parse_args(argc, argv, nx, np);
12
13     if (nx < np)
14         return PMM_INVALID_PARAM;
15
16     /* allocate and initialise data */
17     allocate_gb(gb);
18     gb->nx = gb->ny = gb->nz = nx;
19     gb->np=np;
20     initialize_gb(gb);
21
22     t = pmm_timer_init("dark", pow(nx,3)); /* init timer */
23
24     pmm_timer_start(t); /* start timer */
25
26     /* execute routine */
27     darkmatter(gb->nx, gb->ny, gb->nz, gb->np, ……);
28
29     pmm_timer_stop(t); /* stop timer */
30
31     pmm_timer_result(t); /* get timing result */
32     pmm_timer_destroy(t); /* destroy timer */
33
34     free(gb);
35     return EXIT_SUCCESS;
36 }
```

Figure 3. Example Benchmark Code

In the configuration of PMM the problem developer specifies the parameters to pass to the benchmarking binary, the order that they appear in the function call, the range of each parameter (over which the FPM is to be built) and a path to the binary itself. An example configuration in XML is shown in Fig. 4.

B. Flexible Construction

PMM can construct models in a number of modes. In a GridRPC environment the construction behaviors that are most relevant are those initiated by the administrator of a computing node (rather than by an application, at runtime). Invoked from the command line in an interactive mode, PMM can construct all models it has been configured with at that instant. This provides for accurate model construction before a server is enabled on the Grid, but for a large number of problems, it is a lengthy process that could occupy a machine for an unacceptable amount of time.

For situations where a server cannot be removed from a Grid for the model construction process, PMM can be started as a daemon process. In this mode, the construction of FPMs could be less intrusive. Time constraints, system conditions

```
<problem>
    <name>darkmatter</name>
    <exe_path>/usr/lib/pmm/darkmatter</exe_path>
    <model_path>/var/pmm/darkmatter_model</model_path>

    <parameters>
        <param>
            <name>nx</name>
            <order>0</order>
            <range>
                <min>32</min>
                <max>256</max>
            </range>
        </param>
        <param>
            <name>np</name>
            <order>0</order>
            <range>
                <min>32</min>
                <max>256</max>
            </range>
        </param>
    </parameters>

    <priority>30</priority>

    <benchmarking_policies>
        <policy>
            <time_constraint type="now"></time_constraint>
            <condition type="user_login">
            </condition>
            <condition type="halt_file">
                <halt_path>/tmp/.pmm_halt</halt_path>
            </condition>
        <policy>
    </benchmarking_policies>

</problem>
```

Figure 4. Example configuration of darkmatter problem in PMM

and problem priorities can be applied to manage the building process with a maximum level of flexibility provided to the system administrator.

Time constraints limit the periods when models are permitted to be constructed. Three constraints have been implemented:

- now – construct a model as soon as possible with no time limit on benchmark execution
- until – allow construction of a model up until a certain time, at which point, end construction or allow another time constraint to take over
- periodic – construct a model in specific time intervals, which can be defined by the minute of an hour, the hour of a day, day of week, etc.

Along with each time constraints are halting conditions. These are monitored conditions that can prevent benchmarking. We make a number of conditions available to monitor as well as test for the existence of a halt-file. The halt-file allows an administrator to add any halting condition they wish via an external program that creates and removes the file. The conditions implemented are:

- user login – halt construction if a user is logged into the machine
- load threshold – halt construction when load is above a threshold, this condition is not monitored while a benchmark is being executed
- process detection – halt the construction if a particular process is detected
- user process detection – halt the construction if any process that does not belong to an exclusion list of users is detected
- halt file – as described, a specific file is tested for existence and construction is halted on that basis

Finally we allow each problem to have a construction priority. Problems with a higher priority are constructed to completion before the construction of other problems is begun. Problems with the same priority are scheduled based on their level of completion, always choosing to benchmark a problem that is less complete first.

All constraints on construction can be applied system wide, to all problems configured in PMM, but specific problems can have specific constraints applied to them, which override the system wide configuration. For example, the general timing policy may be that benchmarks are only executed on weekends, but some high priority problem may have a less limiting constraint allowing it's benchmarks to be executed during weekdays provided there are no GridRPC-user processes detected.

When halting conditions are encountered, no benchmarks will start executing until the conditions have cleared. However, if a benchmark is already executing a decision must be made as to whether to allow it to complete or signal it to halt. The action to take is a configurable option. If the halting strategy is to interrupt executing benchmarks and the time constraints / halting conditions are very limiting, lengthy benchmarks may never be run to completion. Consequently some models may never be completely constructed. To mitigate this issue the scheduler takes a number of actions:

1. Benchmarks of a large size are added to the rear of a problem's benchmark queue.
2. Interrupted benchmarks are moved to the rear of a problem's benchmark queue.
3. Repeatedly interrupted problems have their priority reduced.

Though none of these actions prevent this issue entirely, they do delay the point at which it would interfere with FPM construction. Ultimately it is for the administrator to decide how to un-constrain the construction so it may complete.

The design of benchmarking scheduler is trivial. As it has a periodic duty to check the halting conditions, this fixed loop can also be used to schedule new benchmarks. The algorithm is as follows:

- while (1)
  - update system condition data
  - if a benchmark is currently executing
    - if its execution-policy is no longer satisfied
      - halt benchmark, if halt-able
  - else
    - if the global execution-policy is satisfied
      - execute benchmark on top priority problem
    - else
      - for problems with specific execution-policy
        - if the problem's execution policy is satisfied
          - add problem to an executable list
      - execute benchmark for the top priority executable problem in the executable list
  - sleep

C. *Enabling Access and Use of FPM*

PMM provides external programs with access to models in two manners. First, direct access to the FPMs is available via the file system. The PMM API provides methods to locate and parse the FPMs stored on a system into data structures. The API also provides accessor methods to look up an execution time approximated by the model, given a particular set of problem parameters. New points in the FPM can be added to the model when using files, but only if the models are not in the process of being constructed by a PMM process.

When running as a daemon, the manager can service requests for models via socket instead. It accepts the submission of benchmark timing via socket also, which may come from actual executions of a problem that have had timing code inserted. If construction is ongoing when an actual execution time is submitted, the submission can be processed by the GBBP algorithm and can aid in further minimizing construction time. A set of methods is provided for conveniently opening a socket to PMM and sending or receiving data, in the form of individual benchmarks or whole models.

III. GRIDSOLVE AND PMM

The steps involved in a GridRPC call using GridSolve are illustrated in Fig. 5. There are three actors: the client, agent and server. Servers compute problems on behalf of the client. The agent maintains a list of registered GridRPC servers that it may offer to clients. Each server communicates to the agent the problems it can solve and periodically sends an up-to-date performance index for the server. When the client makes GridRPC call it first communicates with the agent, sending a description of the problem it wishes to have solved and in return receives an ordered list of servers ready to service the request. The client then selects a server and sends a request to solve the problem directly to that server.

The list of servers sent to a client is ordered using GridSolve's scheduler on the agent. Amongst other things, the scheduler uses a servers 'score' to decide how to order the server list. For a given problem request, the agent calculates a score for each server that has the ability to compute the problem. The score is a representation of the time that a server would require to execute the problem. This is calculated using two components, a measure of the problem's complexity and a measure of the server's speed.

The problem's complexity is set by the problem developer during its configuration in GridSolve. It is a function of the scalar arguments of the problem, which are known to the agent when it is calculating a server's score. The speed of a server is measured in floating point operations per second using a LINPACK type benchmark. The problem complexity divided by the server's FLOPS gives the server's score.

As previously mentioned, a single benchmark can be a poor representation of a processors speed when the problem being executed is not similar to the benchmarked problem or when the processor uses a different memory hierarchy to the benchmark. Also, the complexity function is something that must be estimated by the problem developer and there is no guarantee that it is in any way accurate.

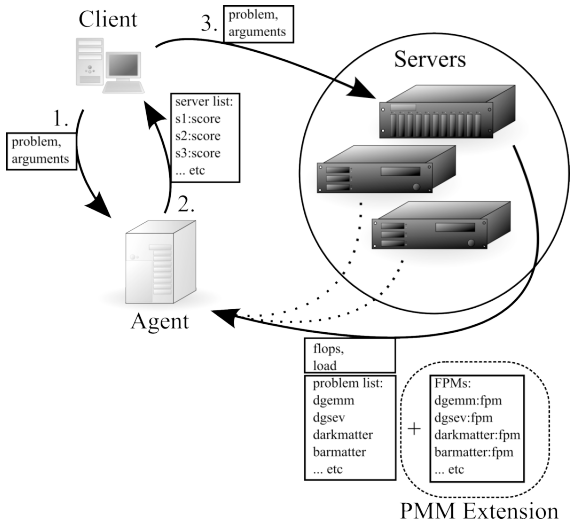

Figure 5. Illustration of Scheduling Transactions in GridSolve

The Functional Performance Model overcomes both of these issues. It models the performance of each specific problem for a range of input sizes, not at only a single point and not using a characteristic application, but the actual problem itself. It is an experimental model, which can render approximate execution times of a problem, without heavily relying on the accuracy of any information that must be provided by to it by the problem developer.

Integration of the FPM in GridSolve requires no fundamental modifications to GridSolve's design. Server scores are a rough estimation of execution time of a problem with a given set of arguments; the FPM provides exactly this, so from the scheduler perspective, no changes are made. No changes are required on the client side either.

The "Smart" extension to GridSolve (SmartGridSolve [8]) uses the same mechanism to retrieve estimations of problem execution time. It has the ability to schedule groups of parallel problems in a single mapping. When scheduling a group of problems the scores of the problems on all available servers are input to a scheduling algorithm. Inaccuracy in the estimation of execution times severely limits the ability of a scheduler in its search for an optimal mapping.

Functionality is added on the server and agent via compile flags set during the configuration of the GridSolve. Where previously, the server would communicate to the agent a list of installed problems at start up, it now must also provide the agent with FPMs for those problems. The server retrieves the FPMs either directly from the PMM daemon via socket or from the file system. The server also submits timing from actual executions to PMM.

Modification to the agent is only in networking code to receive models from the server and in calculating a server's score using the FPM. Common socket code can be used in the server to agent and server to PMM communications. Apart from extending the networking protocol between the agent and server, the majority of the required code permitting

the use of FPM in GridSolve and SmartGridSolve exists in PMM's shared library. No modification to the scheduler is necessary.

IV. HYDROPAD AND PMM

Hydropad [10] is a simulation of the evolution of clusters of galaxies in a universe that is comprised of baryonic matter and dark matter. The core loop of this simulation models the internal interactions of baryonic matter and dark matter, separately and in parallel, while their mutual interaction is modeled in a sequential gravitational calculation. The structure of the application is illustrated in Fig. 6. A GridRPC version ([11]) of this application has been implemented to demonstrate the performance of the Smart extension to GridSolve. Each task in the graph is implemented as a remote procedure call. As a result of data dependencies between time-steps it is not possible to unroll the loop, which limits the level of task parallelism. Further, the volumes of data that must be communicated by the tasks are high. These properties make it particularly challenging for a GridRPC middleware to achieve high performance when running the application. It is for this reason that Hydropad is a good application to examine the performance of GridSolve and the benefit of using FPMs in GridSolve.

The data manipulated by the simulation are three-dimensional cubic matrices that describe the particles in the system (in terms of position, pressure, density, etc). The number of particles in the system is defined by N_p. The accuracy of the overall simulation is determined by the number of cells which the simulation space is divided into. These cells are in a cubic grid structure, the size of which is given by N_x.

The major computational problems in Hydropad are those contained in the main loop. The dark matter problem, *darkmatter*, is a Particle-Mesh N-Body algorithm with a complexity of $O(N_p)$. The baryonic matter, *barmatter*, problem is a Piecewise Parabolic Method with a complexity of $O(N_x)$.

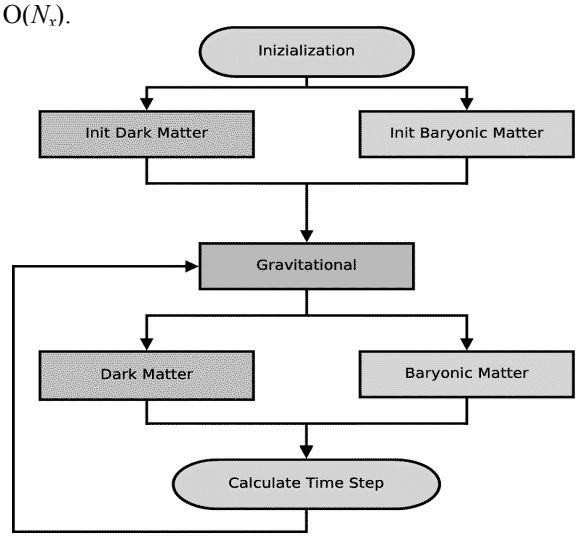

Figure 6. Task Graph of Hydropad Application

In the context of Functional Performance Models, both of these problems are interesting ones. The volumes of data they operate on are different. *darkmatter* takes as input parameters both the particles in the system, specified by N_p and the cells of the grid, specified by N_x. *barmatter* only operates on the cells of the grid structure. Despite this, *barmatter* is computationally more intensive. When executing these tasks on a two of heterogeneous machines it is important to note the volumes of data and the memory available to each processor. A simple performance model will map the computationally large *barmatter* problem to the fastest server. However, if the slower server does not have enough memory to compute the *darkmatter* problem without paging, it may be that overall, the tasks would be executed more quickly if barmatter is mapped to the slower server. This is counterintuitive when the only performance information available is a single benchmark.

FPMs for Hydropad problems have been built using PMM. As can be seen in the task graph, there are a number of problems that must be executed prior to execute *barmatter* or *darkmatter* problems. These are associated with the initialization of the data structures and the calculation of gravitational fields. In the benchmarking binary for a particular target problem, timing functions can be added around any of the problems that the target is dependent on. As such, only two benchmarking binaries were required in building the models for Hydropad, as adequate data for the FPMs of initialization and gravitational problems could be retrieved from the benchmarks of *darkmatter* and *barmatter*.

V. MODELS AND EXPERIMENTS

This section presents the FPMs constructed using PMM for the Hydropad application and experimental results in the speed up achieved through using the FPMs in GridSolve, with the Smart extension. As task parallelism is limited only two servers were used in experiments, their configuration is listed in Table 1. All timed results were remote computations, totally independent of the client.

Experiments were carried out to illustrate the benefit of using FPMs when scheduling a group of tasks. For simplicity experiments are focused on a single iteration of the main loop in Hydropad, the parallel problems: *darkmatter* and *barmatter*. Fig. 7 shows FPMs for the *darkmatter* problem, which are in terms of two parameters N_p and N_x. The models for both servers in the experimental setup are displayed. The change in their relative performance as parameters increase in size is illustrated at the base of the graph. It is clear that paging begins on Hcl02 before Hcl10 and that while the relative performance is fairly constant for smaller problem sizes, it changes dramatically when paging starts. At the

TABLE I. SMARTGRIDSOLVE SERVER CONFIGURATION

Name	Type	MFLOPs	Memory
Hcl10	1.8Ghz Opteron	693.85	1024MB
Hcl02	3.6Ghz Xeon	481.68	256MB

maximum problem parameters "allocate-able" by Hcl02, it is computing at a rate that is twelve times slower than Hcl10, when before it was just slightly slower. This is a property of Hcl02's performance that is not represented by a single benchmark. Fig. 8 reveals greater detail in the region of paging for the *darkmatter* task.

Fig. 9 shows the functional performance models for the *barmatter* task. Again, the differing amounts of available

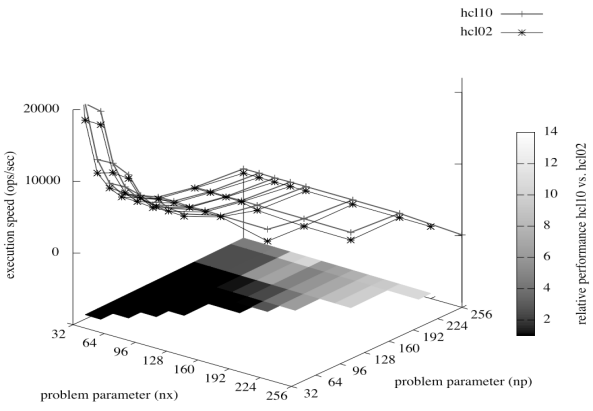

Figure 7. Graph of *darkmatter* Functional Performance Model with relative performance highlighted.

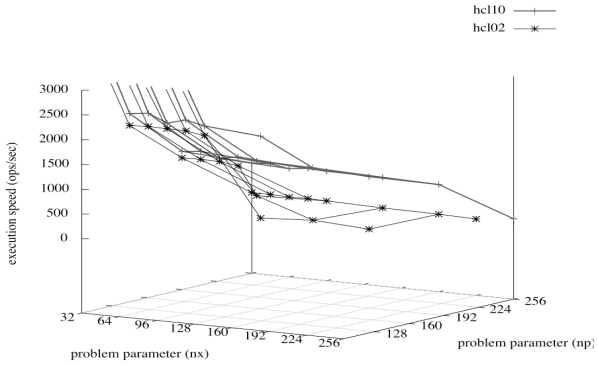

Figure 8. Detail of problem parameters where paging contributes to sudden performance decrease in *darkmatter* problem.

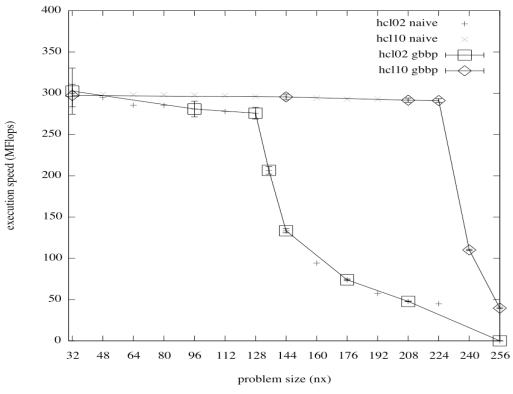

Figure 9. Functional Performance Models for the *barmatter* problem, with points for GBBP and a naïve construction method shown.

memory on the servers results in performance degradation at different values of N_x. Benchmarking points for a naïve construction method are also displayed to illustrate the reduction in the number of benchmarks that are required to build the FPM using GBBP versus a naïve method.

The time spent executing GBBP and naïve benchmarks is shown in Table 2. The speed up achieved by GBBP makes the construction of FPMs a more practical task. In one case GBBP did not achieve a large speed up, the *barmatter* task on Hcl10. This is because an artificial limit was placed on the range of the input value N_x. Had the model been constructed across all "allocate-able" problem sizes, the Geometric Bisection Building Procedure would have shown a consistent speedup.

Finally, Table 3 shows the results of a set of experiments on the scheduling accuracy of SmartGridSolve. A single iteration of the main loop was timed for a set of input parameters. First the scheduler was provided with the standard LINPACK type single benchmark that is made available by GridSolve. As *barmatter* is the most computationally intensive problem, the scheduler assigned it to Hcl10, the fastest server, in all tests. The *darkmatter* problem was executed in parallel on Hcl02. However, when the amount of data *darkmatter* operates on exceeds the available physical memory on Hcl02, it begins to slow. At this point it would be more efficient to assign the computationally intensive *barmatter* to the slower server, as it would be able to solve this problem without paging. The scheduler is not able to make this decision when the performance model of the processor does not represent the change in speed at different levels of the memory hierarchy.

TABLE II. TIME SPENT CONSTRUCTING FUNCTIONAL PERFORMANCE MODEL

Task / Machine	Naïve points	Naïve time	GBBP points	GBBP time	Speed-up factor
darkmatter / Hcl02	76	6854s	36	2292s	2.99
darkmatter / Hcl10	80	1704s	36	598s	2.85
barmatter / Hcl02	14	8792s	7	3687s	2.38
barmatter / Hcl10	15	8262s	6	7444s	1.11

TABLE III. SCHEDULING IMPROVEMENTS WITH FPMS

N_p	N_x	Iteration time: single benchmark	Iteration time: FPM	FPM speed up factor
96	96	26.33s	26.17s	1.01
128	96	25.10s	24.91s	1.01
160	96	25.45s	24.59s	1.03
192	96	41.48s	29.63s	1.40
216	96	123.78s	27.98s	4.42
256	96	n/a	39.02s	n/a
288	96	n/a	51.98s	n/a
320	96	n/a	362.57s	n/a

When the scheduler uses FPMs in its decision-making the results are much better. When no paging occurs, it schedules in exactly the same way as before, but as Hcl02 begins to page it is assigned the problem with the smaller memory footprint. This permits a more optimal scheduling with much greater overall performance. It also allows larger problem sizes to be executed. Previously, after the number of particles, N_p exceeded 256 the *darkmatter* problem failed to execute on Hcl02 as it could not allocate enough virtual memory.

VI. Conclusion

This paper has presented a tool, the Performance Model Manager, which has been designed to enable the construction and use of Functional Performance Models. Its goals are to build the FPM in the most efficient manner possible and to minimize the disruption to a running server. To these ends, it implements the Geometric Bisection Building Procedure and it allows the user to utilize a flexible set of constraints on the benchmarking procedure.

The configuration of PMM and how it benchmarks a problem in order to construct the problem's FPM has been described and models constructed for an example application have been shown.

FPMs can enable more efficient parallel computing in any heterogeneous network of computers. This paper illustrates the use of them via the PMM tool in a Grid environment. Hydropad and SmartGridSolve have been used to demonstrate the application of FPMs in a GridRPC system. The improvement in scheduling can been seen to be significantly more optimal and to permit the execution of much larger problem sizes that were possible with a basic model of processor performance.

Acknowledgment

This research was supported by the Irish Research Council for Science, Engineering and Technology (IRCSET) and IBM under grant RS/2004/IBM/5.

References

[1] H. Nakada, S. Matsuoka, K. Seymour, J. Dongarra, C. Lee, and H. Casanova, "GridRPC: A Remote Procedure Call API for Grid Computing". *Technical report*, Univ. of Tennessee, June 2002. ICL-UT-02-06.

[2] M. Maheswaran et al., "Dynamic Mapping of a Class of Independent Tasks onto Heterogeneous Computing Systems," *Journal of Parallel and Distributed Computing*, vol. 59, 1999, pp. 107—131.

[3] J. Dongarra, "NetSolve: A network server for solving computational science problems," *The International Journal of Supercomputer Applications and High Performance Computing*, vol. 11, 1997, pp. 212—223.

[4] Y. Caniou and E. Jeannot, "Multi-Criteria Scheduling Heuristics for GridRPC Systems", *International Journal of High Performance Computing Applications*, vol. 20, 2005, pp. 61—76.

[5] A. YarKhan, K. Seymour, K. Sagi, Z. Shi, and J. Dongarra. "Recent Developments in GridSolve", Y. Robert, editor, *International Journal of High Performance Computing Applications (Special Issue: Scheduling for Large-Scale Heterogeneous Platforms)*, vol. 20. Sage Science Press, spring 2006.

[6] J. Dongarra, P. Luszczek, and A. Petitet, "The LINPACK Benchmark: Past, present, and future," *Concurrency and Computation: Practice and Experience*, vol. 15, 2003, p. 820.

[7] A. Lastovetsky and R. Reddy, "Data partitioning with a realistic performance model of networks of heterogeneous computers,", *International Parallel and Distributed Processing Symposium IPDPS'2004*. IEEE Computer, 2004, pp. 26—30.

[8] T. Brady, E. Konstantinov, and A. Lastovetsky, "SmartNetSolve: High Level Programming System for High Performance Grid Computing,", *Proceedings of the 20th International Parallel and Distributed Processing Symposium*, Rhodes Island, Greece: IEEE Computer Society, 2006.

[9] A. Lastovetsky, R. Reddy, and R. Higgins, "Building the Functional Performance Model of a Processor," *Proceedings of the 21st Annual ACM Symposium on Applied Computing*, Dijon, France: ACM, 2006.

[10] C. Gheller, O. Pantano, and L. Moscardini, "A cosmological hydrodynamic code based on the piecewise parabolic method," *Monthly Notices of the Royal Astronomical Society*, vol. 295, Apr. 1998, p. 519.

[11] A. Lastovetsky, T. Brady, and M. Guidolin, "Experiments with SmartGridSolve: Achieving Higher Performance by Improving the GridRPC Model," *The 9th IEEE/ACM International Conference on Grid Computing*, Tsukuba, Japan: 2008.