Power performance models for parallel applications

Francisco Almeida, Vicente Blanco, and Javier Ruiz[1]

Abstract— **With energy costs now accounting for nearly 30 percent of a data centre's operating expenses, power consumption has become an important issue when designing and executing a parallel algorithm. This paper analyzes the power consumption of MPI aplications following the master–slave paradigm. The analytical model is derived for this paradigm and is validated over a master–slave matrix–multiplication. This analytical model is parameterized through architectural and algorithmic parameters, and it is capable of predicting the power consumption for a given instance of the problem over a given architecture. We use an external, metered, power distribution unit has been that allows to easily measure the power consumption of computing nodes without the needings of dedicated hardware.**

Keywords— **Energy–efficient algorithms; Power performance**

I. Introduction

In recent years power consumption has become a major concern in the operation of large-scale datacenters and High Performance Computing facilities. As an example, the 10 most powerful supercomputers on the TOP500 List(`www.top500.org`) each require up to 12 MW of power for the entire system (computing and cooling facilities). As a result, power-aware computing has been recognized as one of critical research issues in HPC systems.

New processor architectures allow power management through a mechanism called *Dynamic Voltage and Frequency Scaling* (DVFS) where applications or operating system has the ability to select the frequency and voltage on the fly. Depending on required resources for the application you can select a combination of voltage and frequency, denoted as processors state or *p-state*. Different *p-states* deal to different power consumption, allowing power management by applications [1].

Power measurement can give us information about the energy consumed by systems, but is not sufficient for challenges such as attributing power consumption to virtual machines, predicting how power consumption scales with the number of nodes, and predicting how changes in utilization affect power consumption [2]. These tasks require accurate models of the relationship between resource usage and power consumption. Models based on architectural parameters has been widely developed [3], [4].

We have developed an instrumentation framework based on metered PDUs (Power Distribution Units), allowing to measure the power consumption of HPC nodes while applications are executed. In a similar way we model application performance (execution time) [5], [6], we propose to model power consumption using architectural parameters (number of cores, cache misses, memory access, network latency and bandwidth) and algorithmic parameters (problem size). The analytical power models obtained can be used by schedulers to save energy when applications are executed on HPC systems.

The rest of this paper is organized as follows: Section II introduces master–slave paradigm and Section III describes our experimental setup and measurement system. In Section IV we introduce an analytical power model for the proposed application. We show the obtained models on Section V. Finally, Section VI summarizes the paper with some conclusions and future work.

II. The Master Slave Paradigm

Under the Master–slave paradigm it is assumed that the work W, of size m, can be divided into a set p of independent tasks $work_1, ..., work_p$, of arbitrary sizes $m_1, ..., m_p$, $\sum_{i=1}^{p} m_i = m$, that can be processed in parallel by the slave processors $1, ..., p$. We abstract the master–slave paradigm by the code:

```
for (proc = 1; proc <= p; proc++)
        send(proc, work[proc]);
for (proc = 1; proc <= p; proc++)
        receive(proc, result[proc]);
}

/* Slave Processor */
receive(master, work);
compute(work, result);
send(master, result);
```

The total amount of work W is assumed to be initially located at the master processor, processor p_0. The master, according to an assignment policy, divides W into p tasks and distributes $work_i$ to processor i. After the distribution phase, the master processor collects the results from the processors. Processor i computes $work_i$ and returns the solution to the master. The master processor and the p slaves are connected through a network, typically a bus, that can be accessed only in exclusive mode. At a given time-step, at most one processor can communicate with the master, either to receive data from the master or to send solutions back to it. The results are returned back from the slaves as a synchronous round robin.

The transmission time from the master processor to the slave processor i will be denoted by $R_i = (\beta_i + \tau_i w_i)$ where w_i stands for the size in bytes associated to $work_i$ and β_i and τ_i represent the latency and per-word transfer time respectively. This communication cost involves the time for the master to

[1]Dpto. Estadística, I.O. y Computación, Univ. La Laguna, e-mail: `vblanco@ull.es`

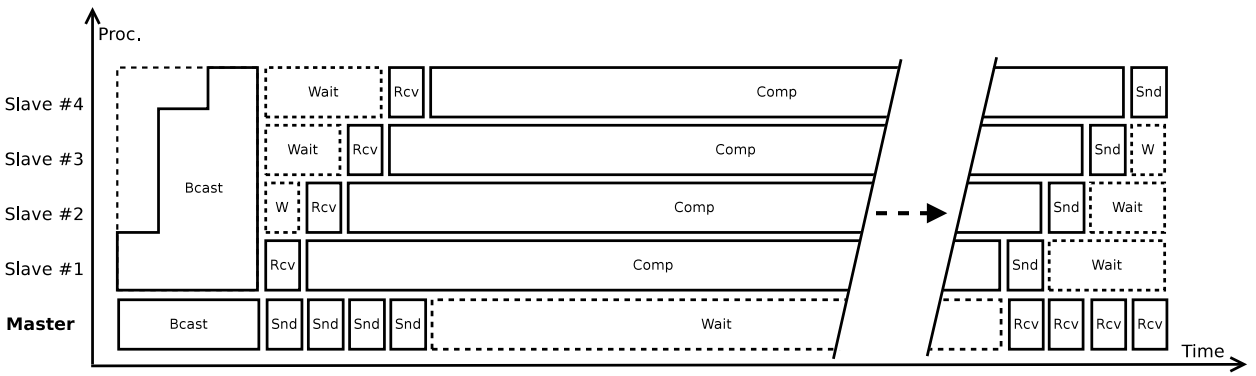

Fig. 1. Master-slave diagram of time for matrix multiplication

send data to a slave and for it to be received. The latency and transfer time may be different on every combination master - slave and they must be calculated separately.

A. Master–slave matrix–multiplication

We introduce the master–slave matrix–multiplication application as a case study. First of all, we describe de algorithm and the corresponding performance analytical model.

For the master–slave model that we are using in this paper (Fig. 1), the time gets approximately reduced by a factor of $1/p$, but a small overload is introduced in the process of broadcasting the matrices and to gather the results. The complete process includes four principal segments: (1) Broadcasting of B matrix, (2) Sending of A matrix by blocks, (3) Waiting for slaves to finish computing and (4) Receiving of matrix C. The total execution time is:

$$T_{par} = T_{bcast} + p \cdot T_{snd} + T_{comp} + T_{rcv} \qquad (1)$$

In order to estimate the time used in the broadcast segment, it is necessary to have determined two parameters related to the network: β_{bcast} and τ_{bcast}, which represent the latency and the bandwidth of the network while performing broadcasting operations. These parameters can be obtained performing a test in C with MPI in the system used [7]. The number of elements to be sent is the size of the matrix squared.

$$T_{bcast} = \beta_{bcast} \cdot \lg(p) + \tau_{bcast} \cdot \lg(p) \cdot n^2 \qquad (2)$$

The send an receive operations performed in the master–slave paradigm can also be modeled in a similar way. Again, two parameters are needed, $\beta_{snd/rcv}$ and $\tau_{snd/rcv}$, obtained with a test referenced in [8]. The first parameter is the latency, but it has a negligible effect, because the sub-blocks to be sent are very few. The τ parameter is the bandwidth in node-to-node communication.

$$T_{snd/rcv} = \beta_{snd/rcv} + \tau_{snd/rcv} \cdot \frac{n^2}{p} \qquad (3)$$

T_{comp} can be modeled by well-known complexity formula $O(N^3)$ for a sequential matrix–multiplication on each processing element. In Section IV-C we give more detail about the model for the computation load and the corresponding energy consumed.

This analytical model to predict the running time of master-slave applications has been widely validated. Our goal now is to obtain a similar analytical model for the power consumption of the master–slave matrix–multiplication algorithm. Section IV describes de model obtained for this implementation on comodity cluster instrumented with a metered PDU.

III. Experimental Setup

The measuring system used is one manufactured by the company Schleifenbauer [9]. The equipment consists of a power distribution unit (PDU) with one input and nine outlets, and a master device called Gateway. The Gateway serves as a hub for the available PDUs. Each PDU is connected to the gateway with standard UTP cable and a RS-485 transport layer at 100 Kbit/s. The Gateway is initially configured through the USB or RS-232 port, assigning it a management IP address. As the Gateway has an Ethernet interface that connects to the appropriate network, this address allows us to access the Gateway through any browser to change its settings, to consult the measurements of any PDU connected and to turn on and off the outlets for each PDU. Gathering of data coming from each PDU is provided through various suitable interfaces (Perl API, HTTP, MySQL, etc.) The setup for the measurements includes the following steps:

1. Setting the IP Gateway with one accessible from our network.
2. Mounting the Gateway and the PDU in the data center.
3. Wire UTP cable from the PDU to the Gateway, and back to the PDU, to create a ring in order to provide for additional redundancy.
4. Connect the Gateway to the network.
5. Connect the AC plugs of the nodes of the system under test to the outlets of the PDU. For convenience is advisable to choose consecutive

output	name	energy total (kWh)	energy subtotal (kWh)	power factor (%)	actual current (A)	peak current (A)	actual voltage (V)
			(...)				
4	node19	[illegible]	[illegible]	97.4	[illegible]	1.25	219.5
5	node20	659	67	96.3	0.81	1.07	219.6
6	node21	716	73	96.4	0.89	1.13	219.5
7	node22	507	66	96.9	0.79	1.04	220.9
8	node23	662	67	97.0	0.80	1.06	220.8
9	node24	815	72	96.7	0.87	1.13	220.8

Fig. 2. Web interface to the Scheleifenbauer Gateway

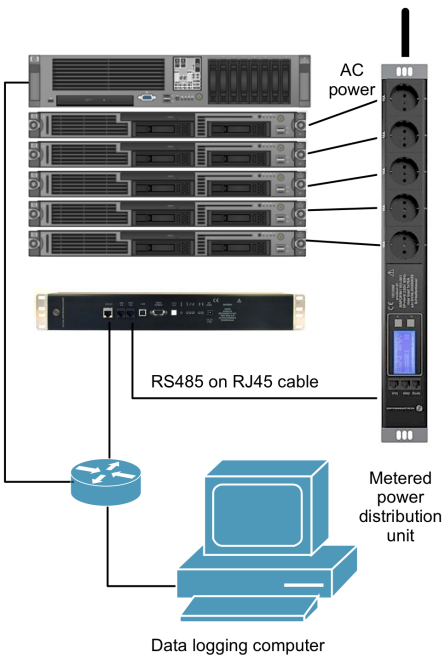

Fig. 3. PDU, Gateway and computing nodes connections

ones.

The equipment installed (Fig. 3) allows the display of consumption data from the PDU by directing a browser to the IP address of the Gateway. The tab "Measurements" allows us to query the data measured in the strip.

In Fig. 2 that shows the web interface, actual current measurements (RMS) for nodes connected to outlets 5 to 9 can be seen. The values correspond to the actual nodes of the cluster which were used in experimentation. Note that the current at idle is not the same for all nodes, although they are identical models. Also it is worth mentioning that the apparent power is the value of current times voltage, expressed in $V \cdot A$. The power factor value tell us how much of the apparent energy is converted into usable energy [10]. Values shown are close to 100%, indicating a good design of the power supplies in the compute nodes.

$$RealPower = V_{RMS} \cdot I_{RMS} \cdot PowerFactor/100$$

The cluster *Tegasaste*, used in the experiment, has 24 nodes, five of which connected to the PDU. The front end node is a 4 x Intel(R) Xeon(TM) @ 3.000 GHz with 1 GB RAM. The computation nodes 20 to 24 have 2 x Intel(R) Xeon(TM) @ 3.200 GHz, 1 GB RAM each. The operating system was Linux 2.6.16.16-papi3.2.1 with gcc version 4.3.2 (Debian 4.3.2-1.1) and MPI 1.2.7.For communication between nodes, an Infiniband(R) switch was used.

A. Measurement

To perform the measurements corresponding to a particular experiment, we use an auxiliary computer, connected to the same network as the Gateway. This auxiliary computer allows the collection of data coming from the PDU, while the algorithm of study is being executed in the parallel cluster, and the logging all the events of interest. The company Schleifenbauer provides an API in Perl to access the measurements. The Gateway has a register type of reading, as current, power factor, voltage, etc.. The user only has to choose the corresponding mnemonic and call the ReadRegisters function. It is important to start this Perl script several seconds before the launching of the execution of the algorithm of study, to take account of the initial values of current in the different nodes. Usually a 10 seconds delay was used. Finally another script executes both the Perl script for the PDU and the parallel algorithm in the CPUs.

The execution of this script generates a set of pairs of files, each pair consisting in one file with PDU data and one with CPU data. As already noted, the idle values of current for each node vary, and so the posterior current measurements will be affected by these idle values. This situation is alleviated by offsetting all the measurements, so they are always zero based. The average idle value of current can be added later on to get the real power consumption. Also, due to the design of the serial protocol connecting the PDU to the Gateway, a measurement takes at least 215 ms. A delay of approximately one second was observed between a measurement and the event that triggers it.

The CPU data file consists in time-stamped events, that can be crossed with the PDU data file to extract the values of current corresponding to each segment of execution. The integration of these cur-

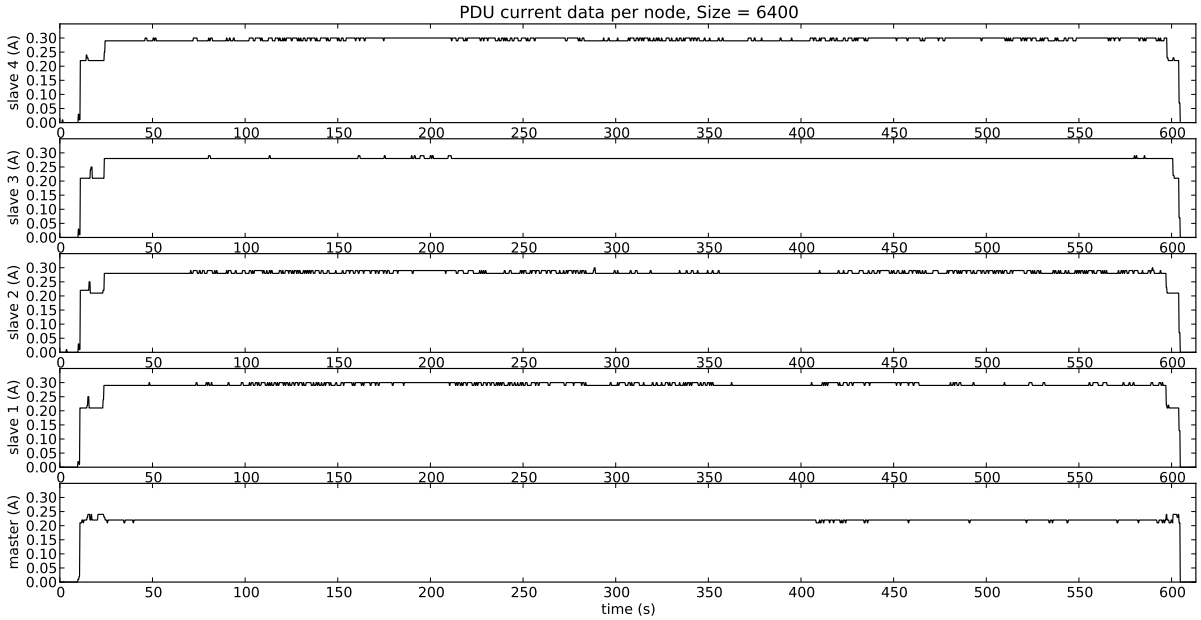

Fig. 4. PDU overall current data, matrix–multiplication on 4 processors

rent values and the corresponding timestamps generates data of measured power consumption of the parallel algorithm.

IV. Power performance model

In this section we propose an analytical power performance model similar to the execution time model introduced in Section II for the execution time of a master–slave application. We will use the well–know matrix–multiplication code to ilustrate the proposed model. First, we study the power–aware behavior of three different sequential matrix–multiplication implementation. Following, we will study the communication part of the master–slave application in terms on power consumption.

A. *Master-slave matrix–multiplication parallel implementation*

To implement the transposed variant of the matrix multiplication we used a master-slave schema, with one master and p slaves, where the master process does not compute. Although each node had two processors, only one was used. The main script of experimentation contained sizes from 1024 to 6400, and every multiplication was performed 10 times. To minimize effects related with the order of execution, this was randomized. Figure 4 shows the overall current profile for a 6400 by 6400 multiplication. The study of the resulting 260 pairs of files with PDU and CPU data allowed us to derive some facts.

- Current kept constant during the broadcast, send and receive segments of code, but its level was higher than that of idle state.
- The current of the master process kept constant at roughly the same level while the slaves performed the computation. We did not find any difference between this level and the one related to the communications segments.
- During the computation segments the current reached its maximum, and again kept constant throughout all the execution.

With all this data available we could estimate the total power consumed by the execution. Taking into account that power consumption is closely related to time, it makes sense to begin with the time model for the particular problem we are studying. From the fragment of code corresponding to the core of the algorithm executed, it could be derived that the time depends on the number of floating points operations $2n^3$ but also from the $2n^3$ accesses to memory (reads) and the n^2 writes to memory.

```
for (i = 0; i < size; ++i){
 for (j = 0; j < size; ++j){
  sum = 0.0;
  for ( k = 0; k < size; ++k){
   sum += A[i * size + k] * B[j * size + k];
  }
  C[i * size + j] = sum;
 }
}
```

To check if the writes to memory affected to the estimated power, a regression with positive coefficients was performed using R [11] and the package *nnls* [12]. It turned out that the write operation did not contribute to the power in this segment. The non zero coefficient obtained, $FlopMem$, includes the contribution of the two floating point operations and the two reads from memory. Thus the estimated time for the sequential case is simply:

$$T_{comp} = 2n^3 \cdot FlopMem \tag{4}$$

The total power of the sequential code is obtained using the average current measured during the execution.

$$PW_{comp} = T_{comp} \cdot Curr_{comp} \tag{5}$$

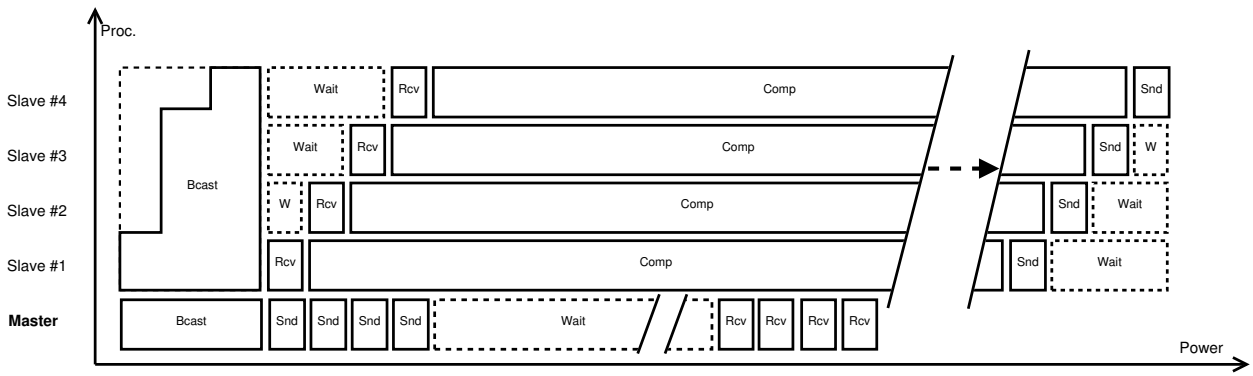

Fig. 5. Master-slave diagram of power consumption for matrix multiplication

B. Model for power consumption

Following the performance model introduced in Section II, we propose a model power compsumption based on equation 1. Each term of this equation contributes with a fraction of the total power consumption, depending on the average current measured in each segment and its duration. Figure 5 depicts again the master-slave schema, this time showing the power consumption. Note the contribution of the master node, lower than the computing nodes (has a lower *p-state*).

At *bcast* operation on eq. 1, it is necessary an average of the current used during the segment, a value that was obtained with the batch of executions, and that it is the same across the processors involved in the operation. The power consumption associated to the broadcast can be computed using:

$$PW_{bcast} = (p+1) \cdot T_{bcast} \cdot Curr_{bcast} \quad (6)$$

where T_{bcast} is modeled by eq. 2.

The power estimation for the send by blocks segment is similar to the broadcast one. With the time needed to send a sub-block of the matrix A, the estimated power for the whole segment can be calculated. A summation expression is used for the sake of completeness and accuracy, although the total contribution of this part is very small.

$$PW_{snd} = (\sum_{i=1}^{p} i + p) \cdot T_{snd} \cdot Curr_{snd/rcv} \quad (7)$$

where T_{snd} is modeled by eq. 3.

The expression for the computation segment includes the p computing nodes and the contribution of the master process while waiting for the slaves to end the computation work:

$$PW_{comp} = p \cdot T_{comp} \cdot Curr_{comp} + PW_{MasterWait} \quad (8)$$

$$PW_{MasterWait} = [T_{comp} - (p-1) \cdot T_{snd}] \cdot Curr_{wait} \quad (9)$$

The last segment of this master-slave consists in the master receiving the results from the slaves. The expression is identical to the send segment one:

$$PW_{rcv} = (\sum_{i=1}^{p} i + p) \cdot T_{rcv} \cdot Curr_{snd/rcv} \quad (10)$$

Finally the complete expression for the power of the parallel execution is:

$$PW_{total} = PW_{bcast} + PW_{send} + PW_{comp} + PW_{recv} \quad (11)$$

Our analysis of the power consumption model must end with a note on units. We have been using $time \cdot Curr$ as a proxy for energy, but this has to be corrected to be completely right. For convenience we have omitted to multiply the current by the voltage, to get $V \cdot A$. Finally we get to the value of $ApparentPower$ in $W \cdot h$ with the appropriate conversion. We have considered the voltage constant during the executions, averaging the voltage of the outlets of the PDU.

C. Sequential matrix–multiplication power consumption

To evaluate power consumption con sequetial codes, we have chosen the matrix–multiplication of dense matrices of size $N \times N$. A dense matrix is a matrix in which most of the entries are non zero. This matrix-matrix multiplication involves $O(N^3)$ operations, since for each element C_{ij} of C, we must compute the dot product of rows with columns.

$$C_{ij} = \sum_{k=0}^{N-1} A_{ik} \cdot B_{kj} \quad (12)$$

With the setup already in place, a test with sequential code was executed and measured with the same sizes than the ones used in the parallel case. Three different variants of the multiplication were used, which we have called standard, transposed and swapped. The standard variant is a word for word translation of the equation (12).

But this naive implementation incurs in many cache misses as the elements of the B matrix are stored in row major format and are staggered in memory. In order to alleviate this source of slowdown, the matrix B is transposed so that memory access will be contiguous for all three matrices. Finally, the swapped algorithm performs partial sums, avoiding the cache problem, but incurring in a greater number of processor operations. Each version has its own current footprint, lower in the case of the

	Measured				Modeled				Error (%)			
Size	$p = 4$	$p = 5$	$p = 6$	$p = 7$	$p = 4$	$p = 5$	$p = 6$	$p = 7$	$p = 4$	$p = 5$	$p = 6$	$p = 7$
2000	25.73	25.452	25.338	24.265	26.14	25.29	24.73	24.33	1.59	-0.62	-2.39	0.27
3000	89.33	85.756	85.833	82.651	88.22	85.37	83.47	82.12	-1.25	-0.45	-2.75	-0.65
4000	212.73	201.985	200.7	196.969	209.11	202.36	197.86	194.64	-1.70	0.18	-1.42	-1.18
5000	412.46	393.303	387.977	383.807	408.41	395.23	386.44	380.16	-0.98	0.49	-0.40	-0.95
6000	714.54	670.201	669.880	656.017	705.73	682.96	667.77	656.92	-1.23	1.90	-0.31	0.14

TABLA I

Power consumption for matrix–multiplication with 4 to 7 slaves, in $A \cdot s$

standard version due to the fact that the processor is waiting for the operands to be fetched from memory. However, the transposed version yielded the minimum energy cost, since it was much faster.

V. Model Validation

The model we have developed depends on architectural and algorithmic parameters that can be measured with the appropriate tests. We show between brackets the values measured in our configuration.

- β_{bcast} [5e-06 s] and τ_{bcast} [4.00641e-09 s], that characterizes the time that the network takes to broadcast a large message.
- $\beta_{snd/rcv}$ [0.0009130886 s] and $\tau_{snd/rcv}$ [1.879013e-08 s], that characterizes the time that takes a node to send a large message to another node.
- $FlopMem$ (1.879013e-08 s] essentially characterizes the computing power of every node, and can be obtained with a simple timed for-loop.
- $Curr_{bcast} = Curr_{snd/rcv} = Curr_{comm} = Curr_{wait}$ [0.2248810 A above idle current] that is the current level at which the communication operations are performed.
- $Curr_{Cmp}$ [0.2921429 A above idle current] characterizes the maximum current per node when one processor is being used at full computation rate.

Finally, table I shows values for measured and modeled matrix multiplications with 4 to 7 slaves. Column labeled Error shows the relative error made by the prediction. The very low error observed, where highest error made is -1.7, allows to conclude that our analytical model has been validated and predicts the power consuption.

VI. Conclusions

We have analyzed the power consumption of the master–slave paradigm over an MPI application. Similar to the performance model in terms of execution time that can be obtained for these kind of implementation, it is possible to obtain an analytical expresion for the energy consumed by these codes while executed on HPC systems. We have implemented a power metered framework based on standard metered PDUs. The experimental infraestructure allows us to monitorize and model any application that can be executed in our cluster. As a case study, we model the matrix–multiplication algorithm by an analytical formula. With this expression we can predict the power consumed by the application on our cluster knowing the problem size and number, the number of slaves used and a set of parameters, architectural–dependent.

Acknowledgements

This work was supported by the Spanish Ministry of Education and Science through TIN2011-24598 and TIN2008-06570-C04-03 projects and through the FPU program. It also was supported by the Canarian Agency for Research, Innovation and Information Society under contract ProID20100222 and has been developed in the framework of the European network COST-ICT-0805 and the Spanish network CAPAP-H2.

Referencias

[1] Min Yeol Lim, Vincent W. Freeh, and David K. Lowenthal, "Adaptive, transparent cpu scaling algorithms leveraging inter-node mpi communication regions," *Parallel Computing*, vol. 37, no. 10-11, pp. 667–683, 2011.

[2] John Davis, Suzanne Rivoire, Moises Goldszmidt, and Ehsan Ardestani, "Accounting for variability in large-scale cluster power models," in *The Second Exascale Evaluation and Research Techniques Workshop. Held in conjunction with ASPLOS 2011*, 2011.

[3] Vincent W. Freeh, David K. Lowenthal, Feng Pan, Nandini Kappiah, Robert Springer, Barry Rountree, and Mark E. Femal, "Analyzing the energy-time trade-off in high-performance computing applications," *IEEE Trans. Parallel Distrib. Syst.*, vol. 18, no. 6, pp. 835–848, 2007.

[4] Laura Keys, Suzanne Rivoire, and John D. Davis, "The search for energy-efficient building blocks for the data center," in *ISCA Workshops*, Ana Lucia Varbanescu, Anca Mariana Molnos, and Rob van Nieuwpoort, Eds. 2010, vol. 6161 of *Lecture Notes in Computer Science*, pp. 172–182, Springer.

[5] D.R. Martínez, J.L. Albín, T.F. Pena, J.C. Cabaleiro, F.F. Rivera, and V. Blanco, "Using accurate AIC–based performance models to improve the scheduling of parallel applications," *Journal of SuperComputing*, vol. 58, no. 3, pp. 332–340, 2011, In press: Online `http://dx.doi.org/doi:10.1007/s11227-011-0589-1`.

[6] D.R. Martinez, J.C. Cabaleiro, T.F. Pena, F.F. Rivera, and V. Blanco, "Performance modeling of mpi applications using model selection techniques," in *18th Euromicro Conference on Parallel, Distributed and Network–based Processing. PDP2010*, Marco Danelutto, Julen Bourgeais, and Tom Gross, Eds., Pisa, Italy, Feb. 2010, pp. 95–102, IEEE Computer Society.

[7] Chau-Yi Chou, Hsi-Ya Chang, Shuen-Tai Wang, and Shou-Cheng Tcheng, "Modeling message-passing overhead on nchc formosa pc cluster," in *GPC*, Yeh-Ching Chung and José E. Moreira, Eds. 2006, vol. 3947 of *Lecture Notes in Computer Science*, pp. 299–307, Springer.

[8] Zhiwei Xu and K. Hwang, "Modeling communication overhead: Mpi and mpl performance on the ibm sp2," *Parallel Distributed Technology: Systems Applications, IEEE*, vol. 4, no. 1, pp. 9 –24, spring 1996.

[9] Alain Schuermans, "Schleifenbauer products bv," Mar. 2012.

[10] Standard Performance Evaluation Corporation, "Spec power and performance, benchmarl methodology v2.1," Aug. 2011.

[11] Various, "R, language and environment for statistical computing and graphics," Mar. 2012.

[12] Katharine M. Mullen and Ivo H. M. van Stokkum, "nnls: The lawson-hanson algorithm for non-negative least squares (nnls)," Apr. 2010.