Proceedings of the Federated Conference on Computer Science and Information Systems pp. 855–864
DOI: 10.15439/2016F308
ACSIS, Vol. 8. ISSN 2300-5963

Modeling energy consumption of parallel applications

Paweł Czarnul, Jarosław Kuchta, Paweł Rościszewski
Faculty of Electronics, Telecommunications and Informatics
Gdansk University of Technology
Narutowicza 11/12, 80-233 Gdansk, Poland
Email: {pczarnul,qhta}@eti.pg.gda.pl, pawel.rosciszewski@pg.gda.pl

Jerzy Proficz
Academic Computer Center
Gdansk University of Technology
Narutowicza 11/12, 80-233 Gdansk, Poland
Email: jerp@task.gda.pl

***Abstract*—The paper presents modeling and simulation of energy consumption of two types of parallel applications: geometric Single Program Multiple Data (SPMD) and divide-and-conquer (DAC). Simulation is performed in a new MERPSYS (Modeling Efficiency, Reliability and Power consumption of multilevel parallel HPC SYStems using CPUs and GPUs) environment. Model of an application uses the Java language with extensions representing message exchange between processes working in parallel. Simulation is performed by running threads representing distinct process codes of an application, with consideration of process counts. Instead of running time consuming calculations, their times are simulated using functions representing computational time dependent on input data sizes. The simulator considers performance and power consumption values for compute devices stored in its database. We performed verification of running the two applications on up to 512 and 1024 processes respectively on a large cluster from Academic Computer Center in Gdansk demonstrating a high degree of accuracy between simulated and measured results.**

I. Introduction

IN TODAY'S High Performance Computing (HPC) landscape performance and power consumption are key factors, both of which are of key concerns in design of future systems. As of today, Tianhe-2 is the most powerful computing cluster on the TOP500 list with performance of over 33 PFlop/s at 17.8 MWs of power consumption. Tianhe-2 uses the hybrid architecture that couples multicore CPUs and accelerators within a single node. Examples of accelerators used today are GPUs or coprocessors such as Intel Xeon Phi. These are used in the top high performance clusters listed on the TOP500 list. The recently announced Tesla P100 offers 5.3 TFlop/s double-precision performance with Thermal Design Power (TDP) 300 Watts[1]. Intel® Xeon Phi™ Coprocessor 7120P offers 1.2 TFlop/s theoretical peak double-precision performance[2] with TDP 300 Watts[3].

As computational power of HPC systems comes from engaging more and more processing cores and consequently increasing the sizes of compute devices and the number of compute devices within a system, there is often a need for assessment of not only performance but also power consumption of such systems. A typical use case is when the user or system owner already know several applications that are run in their contexts or environments and need to assess performance and power consumption of an HPC system after an upgrade or after a new HPC system is to be purchased.

This paper focuses on a model and methodology for assessment of power and energy consumption of parallel applications adopted in the MERPSYS simulation environment[4]. This work follows modeling execution time of parallel applications in MERPSYS that is presented in [1].

II. Related Work

In terms of applications, energy consumption and its reduction is very important. Proper techniques involving load shifting and machine management may result in energy bill savings [2]. Paper [3] analyzes optimization of energy consumption for large virtualized service centers.

In work [4], authors present a workflow that allows prediction of energy and power consumption of HPC applications using available data for a given application regarding power and energy consumption for specific values of nodes used. Then, based on a predictor, that uses the available data and proper interpolation, predicted values can be found. The paper shows a high degree of accuracy of the predictor for Hydro (computational fluid dynamics) and EPOCH (plasma physics simulation) benchmarks executed on the SuperMUC (near Munich, hence MUC) HPC platform.

In paper [5], experiments with Co-Design Molecular Dynamics (CoMD) and Livermore Unstructured Lagrangian Explicit Shock Hydrodynamics (LULESH) codes were performed on a system with host Xeon E5 CPUs and Xeon Phi 5110P coprocessors with measurements of energy and power for the whole system, CPUs and Xeon Phis. Results were used to obtain parameters of theoretical model coefficients with high confidence (R^2 coefficient). Results were presented for host frequency scaling as well as problem size scaling.

In paper [6] authors used neural networks to train models that predicted power and energy consumption when running high performance computing codes. It has been shown that

[1]http://wccftech.com/nvidia-pascal-gpu-gtc-2016/

[2]http://www.intel.com/content/www/us/en/benchmarks/server/xeon-phi/xeon-phi-theoretical-maximums.html

[3]http://ark.intel.com/products/75799/Intel-Xeon-Phi-Coprocessor-7120P-16GB-1_238-GHz-61-core

[4]http://merpsys.eti.pg.gda.pl/

after training, using various versions of codes, it is possible to predict power consumption and energy usage of CPUs and DIMMs with less that 5.5% error for LU factorization, Jacobi and matrix multiplication.

In work [7] authors have presented a detailed energy usage model for parallel master-slave applications, including modeling energy consumption of communication operations, based on execution times. Furthermore, the model was verified in a real environment with a master and 4 or 6 slaves for single or dual core configurations with error rate lower than 4% across the tested configurations.

In paper [8] authors investigated execution times and energy used when running MPI-only and hybrid MPI with OpenMP codes such as Parallel Multiblock Lattice Boltzmann or Gyrokinetic Toroidal Code. In particular, on the largest configurations tested with 8 nodes and a total of 32 cores, hybrid versions showed better execution times and energy consumption than MPI-only codes. Energy used was broken into CPU, memory, disk and motherboard energy.

In paper [9] authors, following analysis performed for Amdahl's law, present a general energy speed-up model in a parallel environment, for multicore systems. Furthermore, authors present specific results for three various power consumption models for a multicore CPU, based on the number of cores used: in the first one all cores are always on, the second with consideration of active cores only and the third with base power, active and idle core power values.

Modeling power consumption of cluster nodes depending on the number of threads active with verification against real measurements were presented in [10]. This showed an idle system power consumption and a non-linear increase until a saturation point. Such a model has been incorporated into the MERPSYS simulator. In work [1], modeling and verification of performance of parallel applications in MERPSYS was presented for computation of vector similarities along with verification in a real parallel environment.

For some types of applications, such as embarrassingly parallel ones, volunteer computing may be an alternative to clusters. Clusters and volunteer systems are different in terms of locality (centralized vs distributed), payer of infrastructure and electrical bill cost, involvement (or lack thereof) of society, security. Comparison of performance and power consumption as well as computational efficiency of cluster based systems and volunteer based systems which use distributed volunteers' computers is presented in [11]. For the latter, sets of volunteer hardware configurations were taken from BOINC projects and `http://cpubenchmark.net/` benchmarks for relevant CPUs and TDPs were used. On average performance/power consumption ratio for modern CPU based clusters turned out to be 2-3 times better than for machines in volunteer based systems.

In [12] authors statistically analyze average CPU utilization and draw a conclusion that in the typical operating region of between 10 and 50% of utilization, energy efficiency is low and aim at designing energy proportional machines that would consume energy proportional to the executed work.

Modelling energy consumption of distributed systems can be useful for exploring the time-energy trade-off, defined in [13]. The authors consider the relationships between execution time, energy consumption and power draw for a set of chosen applications, both on shared memory devices such as Intel Xeon Phi coprocessor and Intel Xeon processor, as well as the Vesta IBM Blue Gene/Q cluster. Formal formulation of the multi-objective code optimization problem is presented, as well as evidence that the energy-performance trade-offs exist in practice.

Paper [14] analyzes energy and makespan trade-offs as a Pareto front in heterogeneous computing systems. Pareto fronts for the multi-objective optimization problem can be determined using mathematical modeling and linear programming [15]. However, such model has to closely match the characteristics of the real executions, which can significantly vary depending on the application model (i.e. synchronization scheme, communication overlapping). Additionally, the model may require defining the execution times of each type of task on each type of hardware beforehand. Thus, for more accurate modeling of various application executions on various hardware, it is important to develop a more flexible method which can give an approximate result with a possibility to quickly modify the application and hardware models.

Paper [16] considers tuning of application execution by proper tiling in the code (cache usage) and CPU frequency. It considers impact on the execution time and energy usage using an example of Poisson's equation with stencil computations.

In work [17] a methodology and experiments were presented for a distributed KernelHive [18] system that is aimed at parallelization of computations in a heterogeneous environment that consists of potentially several clusters each with multicore CPUs and accelerators such as GPUs. Based on an imposed power consumption limit, an optimizer is able to select compute devices such that the total power consumption does not exceed the threshold and execution time is minimized taking into consideration application configuration (including OpenCL's kernel NDRange configurations for GPUs and CPUs), network parameters etc. Dependence of execution times against power consumption limits were shown for a real environment.

As demonstrated in paper [4], a model for prediction of power and energy usage in an HPC system can potentially be very desirable e.g. for budget estimation and prediction of peak requirements in terms of power consumption.

In work [19] authors presented Energy Efficient Task Duplication Schedule (EETDS) algorithm with a grouping and energy efficient group allocation schedule phases of a DAGs (Directed Acyclic Graphs) onto a parallel environment. The algorithm was compared, in terms of energy consumption, to Task Duplication Scheduling algorithm (TDS), Non-Duplication Scheduling algorithm (NDS) and Energy-Efficient Non-Duplication Scheduling (EENDS) strategies for Gaussian Elimination and FFT for various values of communication to computation ratio (CCR) demonstrating benefits of EETDS for larger CCR values.

In paper [20] optimization of hybrid MPI/OpenMP parallel application execution is considered in terms of execution efficiency. Algorithms used consider Dynamic Concurrency Throttling (DCT) and Dynamic Voltage Frequency Scaling (DVFS), also in a combined setting. It is demonstrated that the proposed approach results in savings in energy usage with little loss in performance or even gains.

III. Motivations and Problem Statement

Motivations for simulation of execution of parallel applications on large systems stated for the MERPSYS environment, involving execution time and energy consumption, include:

1) Finding good configurations for running parallel applications i.e. specific compute devices in the available environment, numbers of nodes as well as application parameters such as data packet sizes etc.
2) Testing various potential (e.g. from a database of available components) hardware configurations for running a set of applications. MERPSYS allows instant substitution of one component by another e.g. exchanging a CPU or a GPU for another CPU or GPU model, similarly for interconnects.
3) Simulations of a set of applications in a distributed multi-level system composed of clusters and volunteer based systems in order to find approximately optimal hardware allocation, task mapping and scheduling.

In view of the aforementioned works and challenges, *the goal of this paper is to define and verify a model of energy consumption of a parallel application run on a parallel system that would return acceptably accurate results from fast running simulations of parallel runs*. Specifically, this requires finding the following function

$$\text{energy consumption}(\text{parallel application}, \text{parallel system}, \text{input data})$$

It should be noted that there are two possible ways of how energy consumption is calculated. In one, within the makespan of the application only energy used for duration of computations on particular nodes, only when these are used by the application, is accounted for. In the latter, energy of all nodes is integrated over the makespan irrespective of how many application processes/threads run there, considering idle energy consumption if none processes/threads are active. MERPSYS adopted the second method.

In essence, the function mentioned above can be expressed in terms hardware count, thread count, time of effective application execution (stress time) and time of ineffective processor work (idle time) as follows:

$$\begin{aligned}\text{energy consumption}(\text{parallel application}, \text{parallel system}, \\ \text{input data}) = \\ \sum_{i=1}^{Hardware_{count}} (t_{application} PW[i]_{idle} + \\ \sum_k t_{exec}[i,k](PW[i]_{stress}(threadcount[i,k]) - \\ PW[i]_{idle}))\end{aligned}$$

which considers hardware used and power consumption in idle state multiplied by execution time as well as additional power consumption under stress when running a given number of threads on particular hardware multiplied by activity period.

IV. Modeling Energy Consumption

We modeled energy consumption in a supercomputer Galera Plus located in Academic Computer Centre in Gdansk (CI TASK). This supercomputer consists of 192 computational nodes each containing two Intel Xeon Six-core processors. We used two models of parallel applications: a Single-Program-Multiple-Data application model and a Divide-And-Conquer application model.

Before energy modeling, we modeled the time of a application execution dependency on the number of processors used for calculation. We proved that our timing model is valid using MERPSYS simulation environment (described in the next section). In our simulation, we assumed usage of 1, 8, 27, 64, 125 ... 512 processes for the SPMD application and 1, 2, 4, 8, ... 1024 processes for the DAC application. We achieved results of modeling in a high accordance to the real execution times (see Figure 1) [21].

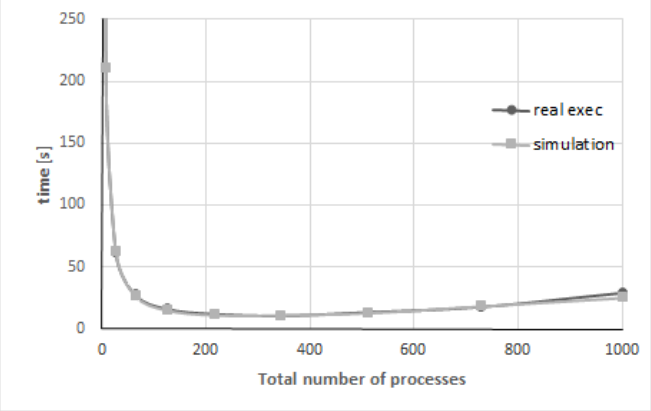
In the first application, all used computational nodes are almost equally loaded during the whole time of application execution. So energy consumption should be a simple multiplication of execution time and the power used by computational nodes involved in calculations. However in our testbed environment only 32 nodes were assigned to experiments. We assumed in our model that when the modeled number of processes was smaller than 32, each process runs on a separate node, and only a part of computational nodes are used. If the modeled number of processes is equal or greater than 32, the processes are distributed among all the available computational nodes, and all the computational nodes are used. So energy consumption in the SPMD application can be expressed as a sum of energy consumed by active nodes (E_{an}) and inactive nodes (E_{in}):

$$E = E_{an} + E_{in}$$

where the energy consumed by active and inactive node are evaluated as:

$$E_{an} = N_{an} \cdot P_{an} \cdot t_{exec}$$
$$E_{in} = N_{in} \cdot P_{in} \cdot t_{exec}$$

(a)



(b)

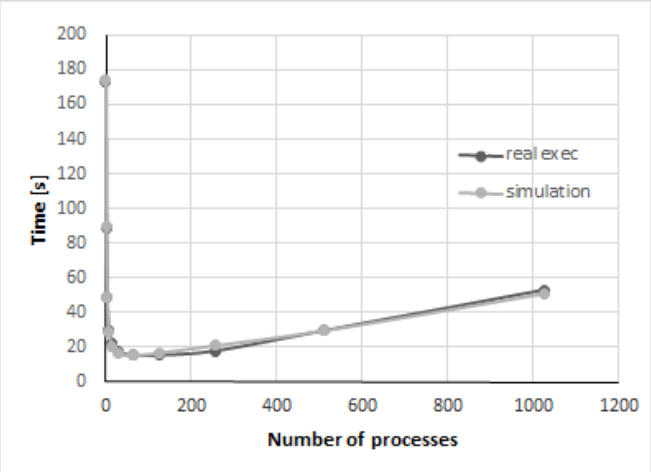


Fig. 1. Execution time modeling of (a) SPMD application and (b) DAC application

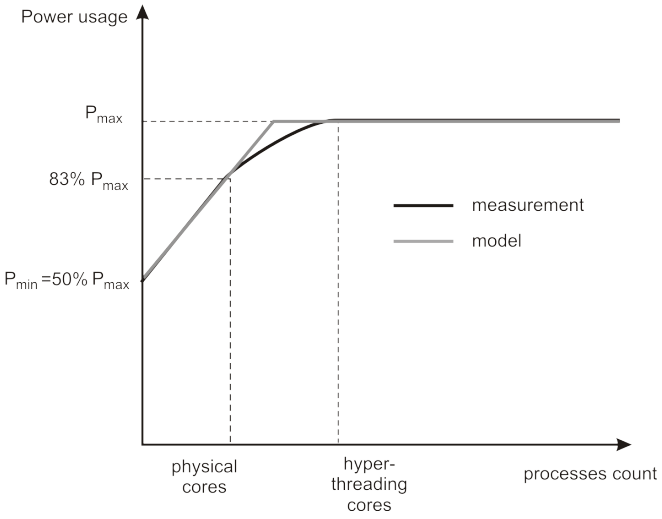


Fig. 2. Power usage on a single node depending on processes count

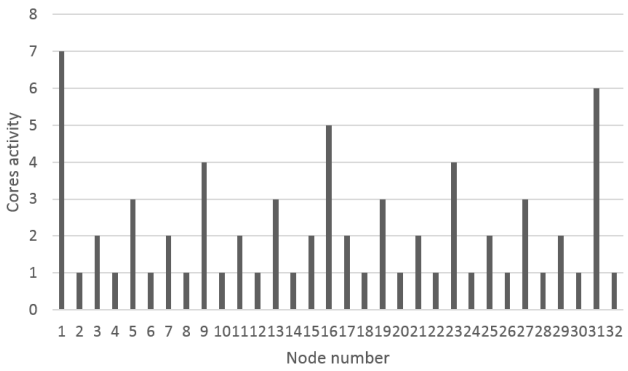


Fig. 3. Model of cores activity in DAC application consecutive steps

The number of active nodes (N_{an}) and the number of inactive nodes (Nin) are simply:

$$N_{an} = min(N_{proc}, N_{total})$$
$$N_{in} = N_{total} - Nan$$

where:

N_{proc} is the number of processes in application,
N_{total} is the total number of computational nodes (here 32),
P_{an} is modeled power usage at one active node,
P_{in} is modeled power usage at inactive node.

We measured that power usage at Intel Xeon processors in inactive node (P_{in}) equals approximately 50% of maximum power usage (P_{max}) which is consumed when all the cores are active [10]. We simplified the function of power usage due to number of active cores as a linear broken function (see Figure 2).

The model of energy consumption in the DAC application is much more complex. Computational nodes are unevenly loaded in consecutive steps of application. At the beginning all needed cores are active. In following steps every second core goes to an idle state (see Figure 3). Thus energy consumption must be evaluated in each step separately. It means that not only the number of active/inactive cores (and active/inactive nodes), but also the time of execution in each application step must be evaluated.

In the DAC application active and inactive energy is expressed by the following formulas:

$$E_{an} = \sum_{k=1..N} E_{an}(k)$$
$$E_{in} = \sum_{k=1..N} E_{in}(k)$$
$$E_{an}(k) = N_{an}(k) \cdot P_{an}(k) \cdot t(k)$$
$$E_{in}(k) = N_{in}(k) \cdot P_{in}(k) \cdot t(k)$$

where:

N is the number of steps in the application process,
k is the index of step,
$t(k)$ is time of execution of the k^{th} step,

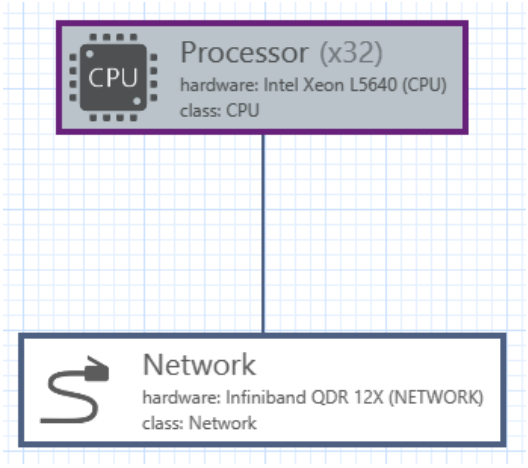


Fig. 4. Sample model of supercomputer architecture in MERPSYS

$P_{an}(k)$is power used on an active node in the k^{th} step,
$P_{in}(k)$is power used on an inactive node in the k^{th} step,
$N_{an}(k)$is number of active nodes in the k^{th} step,
$N_{in}(k)$is number of inactive nodes in the k^{th} step,
$E_{an}(k)$is energy used on all active nodes in the k^{th} step,
$E_{in}(k)$is energy used on all inactive nodes in the k^{th} step,

We are aware that the above model, where the total energy used for computations depends on the number of computational nodes and their usage, ignores the energy used by the whole infrastructure (e.g. cooling system), but this payload was beyond our research at this time.

V. Simulation Environment

We modeled time of execution and energy consumption in the MERPSYS simulation environment. MERPSYS enables modeling of a calculation environment (by drawing an architecture model diagram) and simulation of application execution in this environment (by writing an application model as a simulation program).

The architecture model is a graph diagram in which nodes model key architecture components, and edges model connections between components. As we modeled the Galera Plus supercomputer with homogeneous nodes our diagram consisted of two nodes: one single node modeling all computational components (all processors) and the second node modeling the internal Infiniband network connecting the computational components (see Figure 4). Next we specified component instances count (i.e. the number of computational nodes in the modeled supercomputer). Afterwards MERPSYS looked up to its component database and assigned timing parameters to components.

The application modeling program is written in the Java language, with the use of a special simulator interface, accessible by the `sim` object. We can see a sample fragment of simulation program in Figure 5. The simulation program is not the application itself. To create the simulation program we had to translate the application written in C

```
if (tag.equals("center"))      {

   sim.computation(boundary_cells_size,// computing boundary cells
         ComputationType.CPU,
         SoftwareStack.Undefined,
         1,
         linearComplexity,
         OperationType.Calculations,
         OptimizationType.None);
   sim.p2pCommunicationSend(ysize, "right");
   sim.p2pCommunicationSend(xsize, "bottom");
   sim.p2pCommunicationSend(ysize, "left");
   sim.p2pCommunicationSend(xsize, "top");
```

Fig. 5. Sample fragment of simulation program in MERPSYS

programming language to the simulation Java language. However, the simulation program is much simpler than the corresponding application program. All computational routines are modeled as sim.Computation method invocations. Interprocess communication is modeled as sim.p2pCommunicationSend/sim.p2pCommunicationReceive or one2oneCommunicationSend/one2oneCommunicationReceive invocations. All the researcher has to do is to determine the data count and the computational routines complexity.

VI. Experiments and Results

As we have mentioned above we modeled and simulated time of execution and energy consumption of two parallel applications (SPMD and DAC) in the Galera Plus supercomputer. The applications were written in C using MPI library. We compared the results of simulation with real application execution measured in this supercomputer.

A. Testbed Environment

The testbed consists of a number of identical computation nodes provided by the Academic Computer Center in Gdansk University of Technology in Poland. Each node is based on two Intel(R) Xeon(TM) CPU 2.27GHz processors (EM64T) with 6 physical processing cores with HyperThreading, 12MB cache, running Linux kernel version 2.6.32. Each node has 16 Gigabytes of RAM and they are composed in the cluster architecture, with fast (QDR, 40Gbps) Infiniband interconnection. The power meters of the cluster are served by the specialized, autonomous management subsystems: HP Integrated Lights-Out 3 (iLO 3)[5].

B. Testbed Applications

The first of the tested application is a geometric SPMD application. This kind of application can be used to solve such problems as weather prediction, heat distribution or other physical phenomena. The evaluated 3D geometric space is divided to many cuboidal regions, each region is evaluated by a separate node. The evaluation process is repeated in many iterations, between each iteration the calculation nodes interchanged data corresponding to regions borders. Data is

[5]http://h20565.www2.hpe.com/hpsc/doc/public/display?docId=emr_na-c02714903&lang=en-us&cc=us

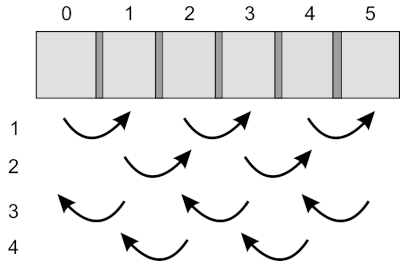


Fig. 6. The schematic inter-process data exchange in the SPMD application

interchanged in 4 steps for each dimension. Considering X dimension in the first step even nodes send data to their right neigbours, next odd nodes send data to their right neighbours, next odd nodes send data to their left neighbours, and at last event neighbours send data to their left neighbours (see Figure 6 for illustration).

Sample fragments of the SPMD application are shown below. In the beginning some common data variables are defined following by four auxiliary routines (getdata, setdata, compute_cell, and cell_to_rank). The main simulation logic is iterated in four nested *for* instructions. The most external loop iterates for an arbitrary number of steps, the internal loops iterate in the three dimensions of the geometric space. After a process has computed an associated cuboidal region in the three dimensions, the process sends data to its neighbors using the scheme shown in Figure 6.

```
double *data;
int X,Y,Z;
int procx,procy,procz;
int proccount;
...
// single data cell get method
double getdata(int x,int y,int z)
{
 return data[...];
}

// single data cell set method
void setdata(int x,int y,int z,double val)
{
 data[...]=val;
}

double compute_cell(int x,int y,int z)
{
 // computes the value of the cell
}

int cell_to_rank(int x,int y,int z)
{
// returns the rank of a process
// that owns cell (x,y,z)
}
```

```
main(int argc,char **argv)
{
 ...
 // the main simulation loop
 for(t=0;t<steps;t++)
 {
  for(i=myminx;i<=mymaxx;i++)
  for(j=myminy;j<=mymaxy;j++)
  for(k=myminz;k<=mymaxz;k++)
  {
   setdata(i,j,k,
           compute_cell(i,j,k));
  }
  // exchange data in X direction
  if (myblockx%2)
  {  // receive from left
    MPI_Recv(..., YZ_wall,
    cell_to_rank(myminx-1,myminy,myminz),
    ...);
  }
  else
  {
   // send to right
   if (myblockx+1<procx)
     MPI_Send(..., YZ_wall,
     cell_to_rank(mymaxx+1,myminy,myminz),
     ...);
  }
  ...
  // do the same for Y direction
  // and Z direction
 }// end of the iteration loop
 MPI_Finalize();
 exit(0);
}
```

The second test application is a Divide-and-Conquer merge-sort algorithm implementation. The first node, which gets the large data set, divides the data into two parts and sends one part to its free neighbor node. This process is repeated in parallel until the size of each partition reaches its limit. Then each node sorts its part of data and "odd" nodes return the sorted fragments to their "parent" nodes. The "parent" nodes merge two sorted data fragments and the process repeats until all data flow to the first node when they are merged to one sorted set (see Figure 7).

The DAC application code is shown below. For simplicity it is assumed that the size of the vector to be sorted is a power of 2. The same applies to the number of processes.

```
...
int *mergelocal(...)
{
// a function for local merging
// (i.e. one process, one thread)
```

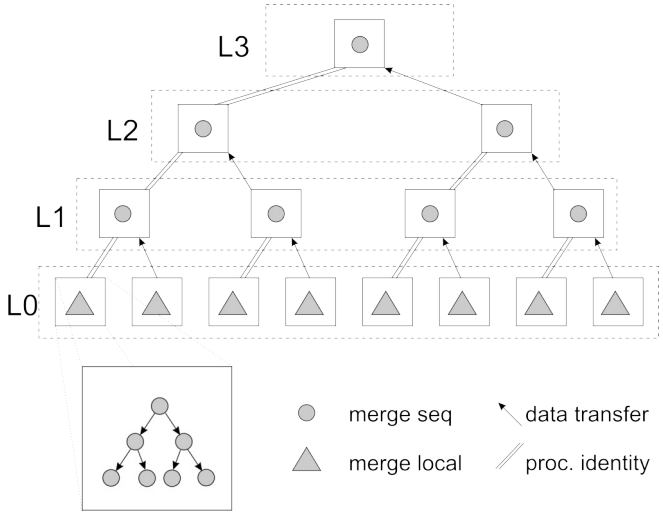


Fig. 7. The schematic algorithm of the SPMD application

```
}

int *mergeseq(int *arrayin,long length)
{
 // the main function for a sequential
 // merge (iterative)
 for(;currentlength*2<=length;
             currentlength*=2)
 {
  for(i=0;i<length;
               i+=2*currentlength)
   mergelocal(...);
   ...
 }
}

int main(int argc,char **argv)
{
 ...
 // in the first step each process needs
 // to sort its part of the array
 arrayout=mergeseq(arrayin, ...);

 // now send the data to an upper process
 if (myrank%2)
 { // then send the data
   // to process with rank myrank-1
   MPI_Send(...);
 }

 // now each process needs to check its
 // role in the divide-and-conquer tree
 // whether it should quit or receive data
 // from another process, merge and send
 // to another process
 int currentskip=2;
 // the current skip between process
 // ranks as in the above scheme
 for(;currentskip<=proccount;
       currentskip*=2)
 {
  if (!(myrank%currentskip))
  {
   // then I am involved in the given step
   // this means that I need
   // to receive the data
    MPI_Recv(...);

   // now merge data
    mergelocal(...);
    ...
   // and send to an upper process
   // if it is not the last iteration
    if (currentskip*2<=proccount)
    { // if not the last iteration
      // now check if I should
      // send the data or not
      if (myrank%(currentskip*2))
      { // then I should send the data
        // to process with rank
        // myrank-currentskip
        MPI_Send(...);
     ...
     break;
    }
   }
  }
 }
 MPI_Finalize();
}
```

C. Simulation Programs

Real calculations are not performed in the simulation program. Instead we invoke sim.computation method passing a string that descibes computational complexity. This string is composed in the simulation program as a JavaScript function that returns the number of operations. However we had to calibrate the result to the real application execution time measured in the testbed environment. It is represented by the last factor (60.94) in the computationComplexity function expression.

```
String computationalComplexity="function "
+ ConstVar.complexityFunctionName + "("
+ ConstVar.parameters + "){"
+ "return " + ConstVar.getDataSize
+"*60.94;"
+ "}";

for (t=0; t<steps; t++)
```

```
{
 sim.computation(compDataSize,
  ComputationType.CPU,
  SoftwareStack.Undefined,
  1,
  computationalComplexity,
  OperationType.Calculations,
  OptimizationType.None);
  ...
```

In the real MPI application each process is identified by a "rank" integer number. In the MERPSYS simulator we can not identify a single process. Instead we can identify a "role" of a process (with a "tag" string). So we mapped application process algorithm based on individual process number to simulation program algorithm based on process group role. We defined 7 roles for the SPMD application using relative processes position: "Center", "Left", "Right", "Top", "Bottom", "Front", and "Back". We replaced the four-step inter-process data exchange with send-receive simulation to the neighbor processes groups (see below):

```
// first send data to all neighbors
neighborTag = "center";
if (!tag.equals(neighborTag)
    && centerCount>0)
  sim.p2pCommunicationSend(wallSize,
                    neighborTag);

  ...

// then receive data from all neighbors
neighborTag = "center";
if (!tag.equals(neighborTag)
    && centerCount>0)
  sim.p2pCommunicationReceive
                    (neighborTag);
```

For the second application, we defined program roles as "levels" (from L0 to L10). We also had to define three computational complexity functions: InitComplexity, MergeSeqComplexity, and MergeLocalComplexity. Instead a peer-to-peer communication send function we used one-to-one communication send. In peer to peer communication, it is assumed that *all* pairs of processes communicate. In the DAC application *one* process sends data to *one* other process. At the same time a *half* of all the processes in the lower level send data to their corresponding processes in the upper level, so the time of one pair communication must be multiplied by sendersCount/2.

```
for (level=0;level<levelCount;level++)
{
 thisTag = "L"+Integer.toString(level);
 if (tag.equals(thisTag))
 {
  nextTag = "L"
          +Integer.toString(level+1);
  if (level==0)
  {
   sim.computation(...,
   InitComplexity,...);

   sim.computation(...,
   MergeSeqComplexity,...);
  }
  else
  {
   priorTag="L"
       +Integer.toString(level-1);
   sim.one2oneCommunicationReceive(
   priorTag);
   sim.computation(...,
   MergeLocalComplexity,...);
  }
  if (level<levelCount-1)
  {
    int sendersCount=
      sim.getNumberOfProcessesForTag
        (thisTag);
    dataSize = dataCount*4;
    sim.one2oneCommunicationSend
      (dataSize, nextTag, sendersCount/2);
  }
 }
}
```

D. Power Measurement

We measured power usage at a single node of a real execution cluster. Power usage was measured in Watts every 10 seconds of the application execution. We repeated the execution three times for each assumed processes count and then we averaged the measured values. The results are shown in Table I and Table II.

TABLE I
POWER USAGE ON A ONE NODE DURING SPMD APPLICATION EXECUTION

Processes count	Measured probes	Power usage on one node [W]	Standard deviation
8	17	84	0.68
27	5	84	0.00
64	2	93	1.49
125	1	106	0.00
216	1	126	0.00
343	1	147	4.71
512	1	159	4.71
729	1	48	0.00
1000	2	99	43.14

E. Simulation Results and Comparison

Having accurate time simulaton results (see Figure 1), we could base energy simulation on solid foundations. We evaluated energy consumption in the MERPSYS simulator and compared the results to the measured energy consumption.

TABLE II
POWER USAGE ON ONE NODE DURING DAC APPLICATION EXECUTION

Processes count	Measured probes	Power usage on one node [W]	Standard deviation
1	16	86.7	2.14
2	8	87.0	1.15
4	4	87.0	1.00
8	2	87.3	0.94
16	1	87.3	0.94
32	1	86.7	0.94
64	1	98.0	0.00
128	1	113	1.89
256	1	137	1.89
512	2	123	43.97
1024	4	134	41,20

TABLE III
REAL AND SIMULATED ENERGY CONSUMPTION IN SPMD APPLICATION

Proc. count	Active nodes count	Inact. nodes count	Real energy cons. [Wh]	Simul. energy cons. [Wh]
8	8	24	151.20	152.1
27	27	5	45.67	46.6
64	32	0	22.71	22.7
125	32	0	15.13	15.0
216	32	0	13.46	13.2
343	32	0	13.86	14.2
512	32	0	18.97	19.0

However as we measured energy consumption at a one node only, we had to recalculate the measured results according to the model of application. In the SPMD applications, as all nodes assigned to the application are active all the time, it was easy to calculate the energy consumption in the whole experimental environment. We show the compared results in Table III. However in the DAC application nodes are unevenly active. We applied the model of activity shown in Figure 3 and recalculated active and inactive nodes real energy consumption in all the applications steps separately, and next summarized them. As the results depend not only on the really measured energy consumption but on the model of activity as well, we call the results the "modeled" energy. Comparison between modeled energy consumption and simulated energy consumption is shown in Table IV.

TABLE IV
MODELED AND SIMULATED ENERGY USAGE IN DAC APPLICATION (COMPARISON)

Proc. count	Modeled energy usage at active nodes	Modeled energy usage at inactive nodes	Total modeled energy usage	Sim. energy usage
1	4.15	118.85	123.0	123.2
2	4.26	59.30	63.6	63.6
4	4.54	30.10	34.6	34.7
8	5.17	15.61	20.8	20.8
16	6.52	7.47	14.0	14.4
32	9.54	2.41	12.0	11.9
64	8.83	3.10	11.9	11.7
128	10.27	4.01	14.3	13.4
256	15.42	5.40	20.8	16.9
512	25.95	7.76	33.7	27.2
1024	47.11	12.83	59.9	55.9

VII. CONCLUSIONS AND FUTURE WORK

In the paper we presented a way to model parallel SPMD and divide-and-conquer applications within the MERPSYS environment including application and system models. Next, we presented verification of results obtained from the fast MERPSYS simulator against energy consumption that stemmed from consideration of power usage of real cluster nodes. We performed tests and calculations for up 512 and 1024 processes for SPMD and divide-and-conquer applications respectively reaching a high degree of accuracy. This allows to obtain results for these applications for other configurations such as input data sizes with ease without the need for rerunning the real application and much faster than the latter.

The future works should cover wider variety of the software and hardware systems. Different vendors and configurations should be tested as well as new simulated programs.

REFERENCES

[1] P. Czarnul, P. Rosciszewski, M. R. Matuszek, and J. Szymanski, "Simulation of parallel similarity measure computations for large data sets," in *2nd IEEE International Conference on Cybernetics, CYBCONF 2015, Gdynia, Poland, June 24-26, 2015*. IEEE, 2015. doi: 10.1109/CYBConf.2015.7175980. ISBN 978-1-4799-8322-3 pp. 472–477. [Online]. Available: http://dx.doi.org/10.1109/CYBConf.2015.7175980

[2] W. McFadden, A. Nikolich, R. Parpart, and B. Runesha, "Saving on data center energy bills with edeals: Electricity demand-response easy adjusted load shifting," in *USENIX Workshop on Cool Topics on Sustainable Data Centers (CoolDC 16)*. Santa Clara, CA: USENIX Association, 2016. [Online]. Available: https://www.usenix.org/conference/cooldc16/workshop-program/presentation/mcfadden

[3] T. Cioara, I. Anghel, I. Salomie, D. Moldovan, G. Copil, and P. Plebani, "Dynamic consolidation methodology for optimizing the energy consumption in large virtualized service centers," in *Federated Conference on Computer Science and Information Systems - FedCSIS 2011, Szczecin, Poland, 18-21 September 2011, Proceedings*, M. Ganzha, L. A. Maciaszek, and M. Paprzycki, Eds., 2011. ISBN 978-83-60810-22-4 pp. 1005–1011. [Online]. Available: http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=6078295

[4] H. Shoukourian, T. Wilde, A. Auweter, and A. Bode, "Predicting the energy and power consumption of strong and weak scaling hpc applications," *Supercomputing frontiers and innovations*, vol. 1, no. 2, 2014. doi: 10.14529/jsfi140202. [Online]. Available: http://dx.doi.org/10.14529/jsfi140202

[5] G. Lawson, M. Sosonkina, and Y. Shen, "Towards modeling energy consumption of xeon phi." *CoRR*, vol. abs/1505.06539, 2015. [Online]. Available: http://dblp.uni-trier.de/db/journals/corr/corr1505.html#LawsonSS15

[6] A. Tiwari, M. A. Laurenzano, L. Carrington, and A. Snavely, "Modeling power and energy usage of hpc kernels," in *Parallel and Distributed Processing Symposium Workshops PhD Forum (IPDPSW), 2012 IEEE 26th International*, May 2012. doi: 10.1109/IPDPSW.2012.121 pp. 990–998.

[7] F. Almeida, V. B. Pérez, A. C. Pérez, and J. Ruiz, "Modeling energy consumption for master-slave applications," *The Journal of Supercomputing*, vol. 65, no. 3, pp. 1137–1149, 2013. doi: 10.1007/s11227-013-0914-y. [Online]. Available: http://dx.doi.org/10.1007/s11227-013-0914-y

[8] C. Lively, X. Wu, V. Taylor, S. Moore, H.-C. Chang, and K. Cameron, "Energy and performance characteristics of different parallel implementations of scientific applications on multicore systems," *International Journal of High Performance Computing Applications*, vol. 25, no. 3, pp. 342–350, 2011. doi: 10.1177/1094342011414749 Energy. [Online]. Available: http://dx.doi.org/10.1177/1094342011414749

[9] R. Isidro-Ramirez, A. M. Viveros, and E. H. Rubio, "Energy consumption model over parallel programs implemented on multicore architectures," *International Journal of Advanced Computer Science and Applications(IJACSA)*, vol. 6, no. 6, 2015. doi: 10.14569/IJACSA.2015.060635. [Online]. Available: http://dx.doi.org/10.14569/IJACSA.2015.060635

[10] J. Proficz and P. Czarnul, *Parallel Processing and Applied Mathematics: 11th International Conference, PPAM 2015, Krakow, Poland, September 6-9, 2015. Revised Selected Papers, Part II*. Cham: Springer International Publishing, 2016, ch. Performance and Power-Aware Modeling of MPI Applications for Cluster Computing, pp. 199–209. ISBN 978-3-319-32152-3. [Online]. Available: http://dx.doi.org/10.1007/978-3-319-32152-3_19

[11] P. Czarnul and M. Matuszek, *Parallel Processing and Applied Mathematics: 11th International Conference, PPAM 2015, Krakow, Poland, September 6-9, 2015. Revised Selected Papers, Part II*. Cham: Springer International Publishing, 2016, ch. Considerations of Computational Efficiency in Volunteer and Cluster Computing, pp. 66–74. ISBN 978-3-319-32152-3. [Online]. Available: http://dx.doi.org/10.1007/978-3-319-32152-3_7

[12] L. A. Barroso and U. Hölzle, "The case for energy-proportional computing," *Computer*, vol. 40, no. 12, pp. 33–37, Dec. 2007. doi: 10.1109/MC.2007.443. [Online]. Available: http://dx.doi.org/10.1109/MC.2007.443

[13] P. Balaprakash, A. Tiwari, and S. M. Wild, *High Performance Computing Systems. Performance Modeling, Benchmarking and Simulation: 4th International Workshop, PMBS 2013, Denver, CO, USA, November 18, 2013. Revised Selected Papers*. Cham: Springer International Publishing, 2014, ch. Multi Objective Optimization of HPC Kernels for Performance, Power, and Energy, pp. 239–260. ISBN 978-3-319-10214-6. [Online]. Available: http://dx.doi.org/10.1007/978-3-319-10214-6_12

[14] K. M. Tarplee, R. Friese, A. A. Maciejewski, and H. J. Siegel, "Efficient and scalable computation of the energy and makespan pareto front for heterogeneous computing systems," in *Computer Science and Information Systems (FedCSIS), 2013 Federated Conference on*, Sept 2013, pp. 401–408.

[15] ——, "Efficient and Scalable Pareto Front Generation for Energy and Makespan in Heterogeneous Computing Systems," in *Recent Advances in Computational Optimization*, S. Fidanova, Ed. Cham: Springer International Publishing, 2015, vol. 580, pp. 161–180. ISBN 978-3-319-12630-2 978-3-319-12631-9. [Online]. Available: http://link.springer.com/10.1007/978-3-319-12631-9_10

[16] A. Tiwari, M. A. Laurenzano, L. Carrington, and A. Snavely, "Auto-tuning for energy usage in scientific applications," in *Proceedings of the 2011 International Conference on Parallel Processing - Volume 2*, ser. Euro-Par'11. Berlin, Heidelberg: Springer-Verlag, 2012. doi: 10.1007/978-3-642-29740-3_21. ISBN 978-3-642-29739-7 pp. 178–187. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-29740-3_21

[17] P. Czarnul and P. Rosciszewski, "Optimization of execution time under power consumption constraints in a heterogeneous parallel system with gpus and cpus," in *Distributed Computing and Networking - 15th International Conference, ICDCN 2014, Coimbatore, India, January 4-7, 2014. Proceedings*, ser. Lecture Notes in Computer Science, M. Chatterjee, J. Cao, K. Kothapalli, and S. Rajsbaum, Eds., vol. 8314. Springer, 2014. doi: 10.1007/978-3-642-45249-9_5. ISBN 978-3-642-45248-2 pp. 66–80. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-45249-9_5

[18] P. Rościszewski, P. Czarnul, R. Lewandowski, and M. Schally-Kacprzak, "KernelHive: a new workflow-based framework for multilevel high performance computing using clusters and workstations with CPUs and GPUs," *Concurrency and Computation: Practice and Experience*, vol. 28, no. 9, pp. 2586–2607, Jun. 2016. doi: 10.1002/cpe.3719. [Online]. Available: http://doi.wiley.com/10.1002/cpe.3719

[19] Z. Zong, X. Qin, X. Ruan, K. Bellam, M. Nijim, and M. I. Alghamdi, "Energy-efficient scheduling for parallel applications running on heterogeneous clusters," in *2007 International Conference on Parallel Processing (ICPP 2007), September 10-14, 2007, Xi-An, China*. IEEE Computer Society, 2007. doi: 10.1109/ICPP.2007.39 p. 19. [Online]. Available: http://dx.doi.org/10.1109/ICPP.2007.39

[20] D. Li, B. R. de Supinski, M. Schulz, D. S. Nikolopoulos, and K. W. Cameron, "Strategies for energy-efficient resource management of hybrid programming models." *IEEE Trans. Parallel Distrib. Syst.*, vol. 24, no. 1, pp. 144–157, 2013. doi: 10.1109/TPDS.2012.95. [Online]. Available: http://dx.doi.org/10.1109/TPDS.2012.95

[21] P. Czarnul, Ed., *Modeling Large-Scale Computing Systems. Practical Approaches in MERPSYS*. Gdansk University of Technology, 2015. ISBN 978-83-938367-2-7