HAEC-SIM: A Simulation Framework for Highly Adaptive Energy-Efficient Computing Platforms

Mario Bielert[†], Florina M. Ciorba[†§], Kim Feldhoff[†], Thomas Ilsche[†], Wolfgang E. Nagel[†]
[†]Technische Universität Dresden, Germany
Center for Information Services and High Performance Computing (ZIH)
{firstname.lastname}@tu-dresden.de
[§]University of Basel, Switzerland
Department of Mathematics and Computer Science
florina.ciorba@unibas.ch

ABSTRACT

This work presents a new trace-based parallel discrete event simulation framework designed for predicting the behavior of a novel computing platform running energy-aware parallel applications. Discrete event traces capture the runtime behavior of parallel applications on existing systems and form the basis for the simulation. The simulation framework processes the events of the input trace by applying simulation models that modify event properties. Thus, the output are again event traces that describe the predicted application behavior on the simulated target platform. Both input and simulated traces can be visualized and analyzed with established tools. The modular design of the framework enables the simulation of different aspects such as temporal performance and energy efficiency by applying distinct simulation models e.g.: (i) A performance model for communication that allows to evaluate the target communication topology and link properties. (ii) An energy model for computations that is based on measurements of current hardware. We showcase the potential of this simulation by simulating the execution of benchmark applications to explore design alternatives of highly adaptive and energy-efficient computing applications and platforms.

Categories and Subject Descriptors

H.4 [**Information Systems Applications**]: Miscellaneous—*computing platforms, communication applications*; I.6 [**Simulation and Modeling**]: Types of Simulation—*discrete event, parallel*; D.2.8 [**Software Engineering**]: Metrics—*performance measures*

Keywords

HAEC, parallel simulation, discrete event, trace-based modeling, performance modeling, energy modeling.

1. INTRODUCTION

It is well known that computers are large consumers of electricity. Energy efficiency is one of the greatest challenges in information and communications technology. A straightforward way for improving energy efficiency is to reduce the energy consumption of every individual hardware component involved. However, it is equally important to understand how the software can be adapted to the hardware on which it runs and vice versa. Today, computational problems are written in software without real opportunities for generating energy-aware code. Energy-unaware code is mapped onto parallel machines with generic hardware configurations. Computational problems often require complex and problem-specific intercommunication between the parallel tasks. Thus, a highly adaptive hardware system, which can optimize its configuration according to the needs of a software system, can provide a much higher level of efficiency than a non-adaptive hardware system. In addition, application states and hardware states need to be monitored and taken into account during runtime as well. Hence, new ways of controlling energy utilization must be found to carefully balance of needs for performance versus cost of energy.

A novel concept, namely the *HAEC Box* [9], utilizes innovative ideas of optical and wireless chip-to-chip communication. It is explored in the Collaborative Research Center HAEC (Highly Adaptive Energy-efficient Computing)[1]. This concept will allow a new level of runtime adaptivity for future computers, creating a platform for flexibly adapting to the needs of the computational problem. The design of the HAEC Box as a whole relies on individual abstraction models of hardware (e.g., CPUs, links), architecture (e.g., computing nodes, network), and software (e.g., runtime system, code generation) as well as their joint interactions. Detailed models are computationally infeasible in design space exploration environments such as the one required by HAEC. Thus, we propose HAEC-SIM as an integrated simulation environment. Via the use of abstraction models, HAEC-SIM allows to predict the performance and energy costs of the HAEC Box running energy-aware applications.

The integrated simulation workflow proposed here comprises four steps: (1) capturing the runtime behavior of parallel applications on a given test system in the form of discrete

[1] `https://tu-dresden.de/sfb912`

SIMUTOOLS 2015, August 24-26, Athens, Greece
Copyright © 2015 ICST
DOI 10.4108/eai.24-8-2015.2261105

event traces; (2) applying abstraction models implemented in simulation to the input discrete event traces; (3) producing output traces describing the predicted application behavior on the simulated target platform; and (4) visualizing and analyzing the input and simulated discrete event traces. Such an approach is known as trace-based simulation (TBS). TBS is a form of discrete event simulation, since application traces consist of a time-ordered series of discrete events, such as send and receive operations, parallel regions, routine transitions, etc. Each event record in the trace has a timestamp, location information (process, thread), as well as event specific data (e.g., message size or region identifiers). TBS uses a specific and realistic input, which allows for precise and detailed validation of the simulation models. Traces preserve the dynamic parallel behavior and can yield meaningful results even for small changes in the model [21]. Most importantly, traces reduce the exploration of the tremendously large simulation design space to tractable solutions.

Contributions. The presented work makes the following contributions. (i) Describe a new parallel discrete event simulation framework, motivated by the need for hardware-software co-design of a new computing platform and its applications. (ii) Showcase the use of the proposed framework for modeling the communication performance and energy consumption of computations for a well known parallel benchmark.

HAEC-SIM constitutes a design space exploration environment to verify the HAEC Box design ideas and to analyze their impact on performance and energy efficiency at large. Systematic analysis of simulation runs allows specification of the design requirements and optimizations for the development of the HAEC system software and hardware.

The remainder of this paper is organized as follows. Section 2 presents an overview of relevant work from the literature. The proposed simulation framework is presented in detail in Section 3. The usefulness of HAEC-SIM is described via two use cases in Section 4. The conclusions and directions for future work are summarized in Section 5.

2. RELATED WORK

There is a vast amount of literature on systems simulation in general, and on computer systems simulation in particular. A part of the literature concentrates on simulating the computer system architecture, while in parts the focus is on simulating the performance of applications.

BigSim [23] is a parallel trace-based simulator for predicting the performance of Message Passing Interface (MPI) applications on systems larger than those available today. The target systems are based on the PERCS (Productive, Easy-to-use, Reliable Computing System) two-level communication network. BigSim uses direct execution (also known as *replay*) to model the two-level MPI communication. **COTSon** [1] is a parallel simulation infrastructure for modeling clusters of multicore CPU nodes, networking, and I/O. It combines functional simulation for the behavior of devices and software, and timing simulators for the timing of all components. Besides the computing performance, it also simulates the power consumption. **Dimemas** [17] is a sequential trace-based simulator for predicting the performance of parallel MPI or multithreaded applications. The simulation model uses parameters such as relative processor speeds, network bandwidth and latency within and across nodes, the number of input and output links, and the processor scheduling policy. The network model assumes two-level buses. **HeSSE** (Heterogeneous System Simulation Environment) [2] is a sequential TBS environment, wherein the simulation components model the functional or temporal behavior of certain system parts. HeSSE allows the prediction of hybrid parallel application performance on heterogeneous computing systems. **LogGOPSim** [13] is a sequential trace-based simulator used for analyzing the performance of MPI applications on existing computing systems. It implements several LogP-derived network models. **MARS** (MPI Application Replay network Simulator) [8] is a parallel trace-based simulator, supporting several network topologies (based on OMNEST / OMNet++) oriented towards the PERCS network. It provides flexible routing schemes, arbitrary application task placement, and uses direct execution for MPI communications to accelerate the simulation. **PSiNS** (PMaC's open Source interconnect and Network Simulator) [22] is a trace-driven performance modeling and prediction tool for MPI applications, similar to Dimemas. PSiNS includes several built-in communication models (namely simple, resource contention, and PMaC) that can be used to investigate a target system. Each event can use a different communication model. PSiNS uses direct execution for MPI calls. **SILAS** (SImulation of LArge-Scale parallel applications) [11] is a parallel trace-based performance simulator for large scale target systems. SILAS focuses on the effects of fine-grain alterations of application-level behavior with respect to the performance under an identical execution configuration. It uses direct execution for MPI communications. **xSim** (Extreme-scale Simulator) [6] is a performance investigation toolkit that uses lightweight parallel discrete event simulation. The simulation of a parallel application is done on a much smaller system in a highly oversubscribed fashion with a virtual wall clock time. Based on a processor and a network model with an appropriate simulation scalability/accuracy trade-off, performance data can be extracted. xSim supports the use of several network topologies, direct execution (or replay) of computations and messages, and models resource contention and failure.

The proposed simulation framework shares certain similarities with the above approaches: using traces as the basis for simulation, allowing for the simulation to be conducted in parallel, focusing on modeling the performance of existing applications on non-existing platforms, and considering heterogeneous communication links. HAEC-SIM, however, significantly differs from the above approaches as follows: (1) communication is simulated with abstraction models as opposed to direct execution (a form of execution based simulation), hence increasing the simulation accuracy at the expense of increased modeling complexity; (2) the communication models contain multiple levels (on-chip links between cores in one CPU, optical links between CPUs on a single board, and wireless connections between CPUs on adjacent boards) and are characterized by dynamic parameters (e.g., in/active links, bandwidth), as opposed to being two-level and modeled with static parameters; (3) the simulation considers the recorded energy consumption of certain system components contained in the input trace; and (4) the simula-

tion goal is to predict the temporal performance and energy consumption of the hybrid parallel application running on the target computing platform.

3. THE HAEC-SIM FRAMEWORK

3.1 Motivating Platform Characteristics

The HAEC Box [9][7] refers to a new high performance-low energy parallel computing platform. The HAEC Box is the motivating target platform for the design and development of the HAEC-SIM framework. In this architecture, the computing nodes are assumed to consist of 3D stacked processor chips with thousands of 'thin' cores and local memory [19] offering massive intra-node parallelism. The *computing node topology* is assumed to be 3D with $n = 64$ nodes organized as $4\times4\times4$. Several computing nodes (4×4) are placed on a single board and connected using optical waveguides. Four such boards are connected using inter-board high-speed wireless links. Adjacent computing nodes on a single board are connected via *optical links* with a transmission bandwidth of 250 Gbit/s, link latency of 10 ns, and bit error rate of less than 10^{-12}. A 2D torus is assumed for the *optical links topology* of a single board. The total number of physical optical links is $l_o = 4 \cdot n/2 = 128$. Computing nodes of neighboring boards can all communicate with each other via *wireless links* with transmission bandwidth of 100 Gbit/s, latency of 100 ns, and bit error rate of 10^{-8}. The *wireless link topology* is fully connected between adjacent boards. The total number of physical wireless links is $l_w = (4-1)\cdot n^2 = 768$. These are the current design characteristics, but the numbers may still change depending on the requirements of the software and further developments in hardware. When it comes to trade-offs, the simulation can provide viable input on how a change in the hardware characteristics affect the software execution performance and energy consumption.

3.2 Design Goals

An integrated simulation environment needs to satisfy a number of requirements. The design goals of HAEC-SIM are described in the following. *Flexibility* in terms of fine-grain accuracy (to derive the final optimized application and system design) versus coarse-grain accuracy (to move more quickly within the application and system design space). The accuracy granularity is strictly dependent on the temporal granularity and the number of events in the input trace. *Modularity/extensibility* to allow coupling with different and novel models to guide the simulation. *Scalability* in terms of the number of captured events in the input trace and the degree of parallelism within the application. *Correctness* between the theoretical models and their implementation into corresponding simulation modules. *User friendliness* via intuitive input/output visualization and simulation launching process. *Portability* across different platforms (Linux, Mac OS and in the future Windows).

3.3 Design and Implementation

An overview of HAEC-SIM is illustrated in Figure 1. The framework is written in C++11 and is organized into three layers: the *interface layer* for external libraries (with the OTF2xx and Boost.MPI components), the *base layer* (with the source, sink, base modules, and resource manager components), and the *simulation layer* (having simulation modules as components). With the exception of Boost.MPI, all components are developed in house.

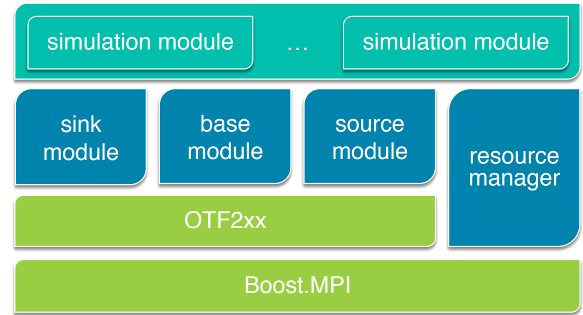

Figure 1: Overview of the HAEC-SIM framework.

The implementation of the three layers depends on other software components: (1) The OTF2 [15] library (version 1.4), which is also part of the Score-P [16] infrastructure (version 1.3); (2) The Boost[2] C++ libraries (version 1.55); and (3) The jsoncpp[3] library, which is distributed with HAEC-SIM.

3.3.1 Input

HAEC-SIM requires three types of input: an input trace, a configuration file, and a mapping file.

Input trace. The input trace represents the execution trace of an instrumented parallel application of interest. The trace contains the desired performance and energy features of the application of interest, in the form of a time-ordered *event* sequence. Each event is assigned a *timestamp*, denoting the time when the event occurred. Each event is also associated with a *location* which denotes where the event occurred. Locations correspond to tasks in a parallel software and in this respect, are generic terms for 'processes', 'threads', 'MPI ranks', 'OpenSHMEM PEs', 'asynchronous metrics', and others. Figure 2 illustrates a parallel event trace as a time-space diagram. The horizontal direction represents time and later times are to the right of earlier ones. The vertical direction represents the space of locations in which different events denoted as circles (with timestamps) occur. The horizontal lines denote locations (application processes/threads, metrics), and the wavy line denotes an application message between processes from different locations.

Metrics capture information about the various components of the system during the execution of the application of interest. The Score-P measurement infrastructure [16] supports two types of metrics: synchronous and asynchronous. Examples of synchronous metrics include PAPI counters[4], while asynchronous metrics include energy measurements. The synchronous metric events have the same timestamp as their associated application events (e.g., function enter and function leave) and are written in the trace of the respective location *before* the application events, as shown in Figure 2. This results in a monotonically increasing (non-strict) or-

[2]http://www.boost.org
[3]https://github.com/open-source-parsers/jsoncpp
[4]http://icl.cs.utk.edu/papi/

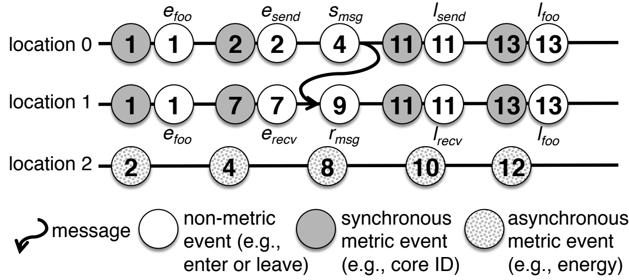

Figure 2: Time-space diagram of events ordered in different locations and their associated timestamps. Function enter or leave events are denoted as e_{func} and l_{func}, respectively. Send and receive message events are denoted as s_{msg} and r_{msg}, respectively.

dering of the events [18]. The asynchronous metrics are by design stored in individual locations due to the fact that the data they represent may not be directly accessible during the execution of the application. Thus, this data is often added postmortem to the trace.

The desired performance and energy features are collected as events using the Score-P measurement infrastructure during the instrumented execution of the application. The events recorded during the running time of the application are stored in the OTF2 format, which is a highly scalable, memory efficient, and binary event trace data format. In OTF2, events are stored as records, which denote the smallest units of trace data. There are two types of records: *definition* records and *event* records. The definition records describe global properties for the entire trace or define IDs to be referenced in event records. Each event record represents a single event. For example, subroutine names (long text strings) are mapped to subroutine IDs (short numbers) in a definition record, and event records reference the subroutine IDs to reduce storage requirements. OTF2 has a fixed record model with a given set of definition/event record types; the data fields of each record type are predefined. In addition, individual records can carry an arbitrary collection of generic attributes via a list of key-value pairs. This allows recording any type of additional data in a most flexible way. An OTF2 trace can be visualized and analyzed using the performance analysis tool Vampir [14]. OTF2 traces are organized as collections of several files: (a) the anchor file (default `traces.otf2`), (b) the global definitions file (default `traces.def`), and (c) a subdirectory (default `traces`) that contains the actual trace data. The subdirectory contains two types of files for each location: (c.1) a local definitions file (i.e., `<locationID>.def`) and (c.2) an event file (i.e., `<locationID>.evt`).

In OTF2, event timestamps are represented as unsigned 64-bit integers with a starting point and a rate (ticks per second) that is defined by definition records. The simulation infrastructure converts the timestamps to 64-bit signed integer picoseconds since the beginning of the trace. Thus, HAEC-SIM can process traces with timestamps up to 2^{63} ps, corresponding to trace lengths of 106 days.

Configuration file. The configuration file (e.g., Figure 3, `haec_sim.conf`) contains parameters related to the simulated system. This includes the topology, as well as parameters required by the selected simulation modules, such as communication link characteristics, or computational resources speed, and others. The configuration file is shared by all simulation modules. Each module's initial configuration is customized into separate sections using the JSON[5] format. The semantic is left to the responsible module, thus allowing for a highly generic and flexible module configuration approach. An example configuration file is listed below.

```
{  "topology": {
    "size" : [4,4,4] },
  "modules": {
    "static_network_model": {
      "communication_model": "PNC",
      "bandwidth": 12500000000,
      "latency": 100,  },
    "power_estimator": {
      "energy_model": "bircher" },
  } }
```

Mapping file. The mapping file (e.g., `positions.map` in Figure 3) contains a map of the process identifiers (e.g., MPI ranks or OpenMP threads) of the parallel application to the computing nodes of the simulated platform topology. The format of the mapping file is exemplified in the listing below for a $4 \times 4 \times 4$ node topology.

```
<mapping name>
x_coord y_coord z_coord number_of_processes process_id(s)
0 0 0 2 38 72
1 0 0 2  3 73
[...]
3 3 3 4 35 51 75 77
```

The first row indicates the name of the mapping strategy used. The strategies available in HAEC-SIM are: xyz (default), block xyz, and random mapping [7]. The third row denotes the fact that the computing node with coordinates (0,0,0) is allocated two locations, and that these locations have IDs 38 and 72 (in this case denoting the MPI ranks of the application processes).

3.3.2 Simulation Core and Modules

Software control flow. The control flow between the HAEC-SIM software components is illustrated in Figure 3. The flow of software control is identical for all simulation processes. The control starts in haec_sim::main, which hands over the control (step 1) to the otf2xx::reader for reading the definitions and events from the input trace. This handing over is called *inversion of control* and is a widely used pattern in software design for increasing modularity and extensibility. The software control flow for reading definitions is slightly different from the one for reading and processing events. For reading and processing the global definitions (i.e., `traces.def`) of the events in the input trace, the control flow steps *once* through steps 2-10 with the exception of steps 4 and 5, as reading the definitions does not require the state of the simulated resources. For reading and processing

[5] JavaScript Object Notation, `http://www.json.org`

the events (i.e., `traces/<locationID>.evt`), the control *repeatedly* flows through all steps between 2-10.

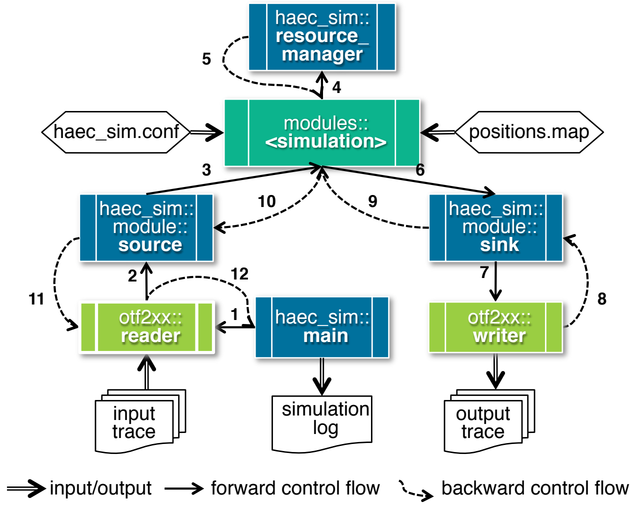

Figure 3: Software control flow in HAEC-SIM.

In step 2, the otf2xx::reader triggers a callback function specific to the definition or event that was read. The callback function, in turn, triggers various methods implemented in the simulation components (e.g., source, <simulation>, and sink) that processing this definition or event (steps 2, 3, [4, 5,] 6, and 7). Events can broadly be classified into computation and communication operations. Depending on the simulation model, in case of a computation event, the state of the simulated computation nodes is probed/updated via control flow steps 4 and 5. Similarly, if the event read is a communication event, the state of the simulated communication links is probed/updated in steps 4 and 5. A simulation module can also delete events by skipping the call to sink or create new events by calling sink multiple times. Once the definition or event has been processed, the control flow returns to otf2xx::reader (steps 8, 9, 10, and 11). When all definitions and events have been read and processed, otf2xx::reader returns the control back to haec_sim::main, which terminates the simulation.

Simulation processes. The simulation runs in parallel and uses $p + n + 2b - 1$ MPI processes, where p is the number of event locations (i.e., the sum of parallel application processes, threads, and asynchronous metrics) in the input trace, n is the number of computing nodes in the simulated platform, and b is the number of boards in the simulated platform (§ 3.1). Each process type has a dedicated MPI communicator, constructed from the default global communicator MPI_COMM_WORLD, as illustrated in Figure 4. Processes in the *processes* communicator operate on the events contained by their assigned locations (e.g., application processes) from the input trace. Processes in the *nodes* communicator maintain the operation state (e.g., active or inactive, power states) and other attributes of the computing nodes (e.g., nominal computing power, available computing power, number of simulated application processes running on it). Processes in the *links* communicator are further split into two communicators corresponding to the optical links and wireless links characteristics, respectively. The links processes maintain the state of the communication links (e.g., active or inactive, power states, characteristics of the messages traveling over the links).

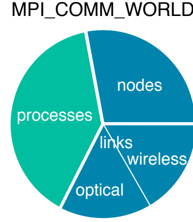

Figure 4: Communicators for the parallel simulation ranks of HAEC-SIM.

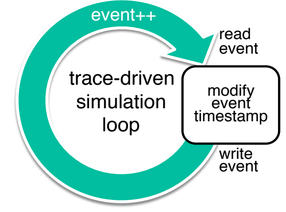

Figure 5: Next-event incrementing time advance method.

The number of event locations, p, depends on the input trace. The codomain of the MPI ranks in the processes communicator is given by $\{0, 1, \ldots, p-1\}$. The number of computing nodes, n, is declared in the configuration file (§ 3.3.1), and represents the product of the 3D node topology description (default $4 \times 4 \times 4$, § 3.1). The codomain of the MPI ranks in the nodes communicator is given by $\{p, p+1, \ldots, p+n-1\}$. These MPI ranks denote the resource managers of the HAEC Box computing nodes. The number of boards, b, is given by the third dimension of the 3D node topology declared in the configuration file (default 4). The codomain of the MPI ranks for the optical links communicator is given by $\{p+n, p+n+1, \ldots, p+n+b-1\}$, while the codomain for the wireless links communicator is given by $\{p+n+b, p+n+b+1, \ldots, p+n+b+(b-1)-1\}$. Each rank from the optical links communicator denotes the resource manager of the links on a single board and manages 32 on-board optical links. Each rank from the wireless links communicator denotes the resource manager of the links between adjacent boards and manages 256 board-to-board wireless links.

If $p \leq n$, then p distinct computing nodes will be allocated p locations, while $n - p$ nodes will remain unallocated. If $p > n$, certain nodes will be allocated more than one location, while other nodes will be assigned each only one location.

Simulation modules. A simulation module exports callback handler functions for the trace record types that it is designed to process. In addition, there are module initialization and destruction functions. Every class implementing a simulation module inherits from haec_sim::module::base. This class provides a default handler for each trace record type. The base class also provides the functionality to automatically modify the timestamps of all events, if the simulation leads to changed timestamps for events of interest.

Multiple instances are created for every simulation module, one for each parallel simulation process p. During the execution of the simulator, each instance processes all definitions and the events contained in one location of the input trace. After processing of each trace record by the simulation mod-

ule, the record is passed onward to the sink module, whereby it is written in the output trace. Each simulation module instance can also pass onward newly created trace records to the sink module. Conversely, a simulation module can discard records from the input trace, and thus they will be absent from the output trace. During processing of event records from the input trace or creation of new records by the simulation module, care must be taken to satisfy the time and causality constraints (e.g., the happens before relation) described in the following Time management paragraph. Thus, the output trace can contain more or fewer trace records than the input trace.

System state. The state of the target platform during the simulation is maintained via resource managers and data stored within the simulation module instances. As resource managers are executed by different simulation processes (the processes in the 'nodes' and 'links' communicators in Figure 4), the simulation module instances and the resource managers must exchange system state information. This communication exchange is explicit and realized using blocking point-to-point MPI messages and collective MPI operations. The system state may refer to event locations mapped to computing nodes, details about application messages traveling over optical or wireless links, and sharing of computing and communication resources. The nodes resource managers are responsible for the sharing policy governing the use of the computing node, while the links resource managers are responsible for the control of the network flows and the shared use of the optical and wireless links. The implementation of resource contention protocols is ongoing.

Parallel simulation. HAEC-SIM is a framework for parallel trace-driven simulation. As shown in Figure 3, for each location there is a main loop (initiated in otf2xx::reader) that calls a general purpose method to read definitions or events from the input trace. Via inversion of control, this method triggers a callback function specific to the definition or event that is currently being read. The simulation time is driven by events. The simulation clock follows the *next-event incrementing* time advance method [12]. Next-event incrementing differs from fixed-time incrementing in that the simulation clock is incremented by a variable amount rather than by a fixed amount each time. This variable amount is the time from the event that has just occurred (was last read) until the time when a subsequent event of any kind occurs (is read); i.e., the clock jumps from event to event. Events are read by advancing the simulation time to the next event of any kind in the input trace (Figure 5). The callback function may (a) modify the timestamp of the last read event, (b) update the state of the simulated system by determining the changes resulting from the modification of this event, and/or (c) call the appropriate methods for writing the (modified) event into the simulated output trace. The framework itself does not imply any synchronization of the time at different parallel simulation processes.

Time management. Synchronization is required to correctly simulate parallel interactions between parallel event locations of the input trace that are processed by different instances of a simulation module [10]. Due to the fact that in parallel simulation there is no shared state among the

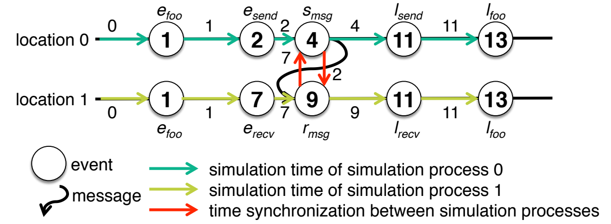

Figure 6: Time synchronization between simulation processes. Function enter or leave events are denoted as e_{func} and l_{func}, respectively. Send and receive message events are denoted as s_{msg} and r_{msg}, respectively.

parallel simulation instances, the interactions between the parallel instances occur via explicit messages. Regardless of the parallelization paradigm used during the recording of the input traces, the simulator uses MPI to provide the necessary synchronization between module instances. MPI was selected due to its standardization and its potential to scale to a large number of processes (exceeding 100,000 ranks). Synchronization between simulation processes is conducted via blocking point-to-point (e.g., time synchronization in Figure 6) or collective communication operations (e.g., initialization of the resource manager processes with mapping information).

The simulator satisfies causality in the simulated parallel program execution by processing the events in their timestamp order across the entire simulation. This means that earlier events may influence later events, while the reverse is not true. Computation events (e.g., e_{foo}, l_{foo} in Figure 6) processed by a single simulation process directly influence only subsequent events from the same location (*local causality*). This can be extended by a resource manager to model shared computation resources. Communication events (e.g., e_{send}, s_{msg}, l_{send}, e_{recv}, r_{msg}, or l_{recv} in Figure 6) may influence events processed by other simulation processes (*global causality*). It follows that computation events can be simulated within the local simulation instance, whereas communication events may require synchronization between two or more simulation instances.

HAEC-SIM employs a *conservative synchronization* (time management) algorithm by avoiding violation of the local and global causality constraints. Specifically, we employ a synchronous algorithm that waits until the timestamps of the next relevant events are known. Synchronization between *processes* simulation instances (see Figure 4) occurs only upon encountering communication events in the input trace. Deadlocks between these *processes* simulation instances are avoided provided that the input trace is valid in terms of communication operations. Synchronization between *nodes*, *links*, and *processes* is required for any simulation that accounts for the state of the simulated system, as the shared access and use of computing nodes and communication links affects the system state. These dependencies must never be circular to avoid deadlocks. This is ensured by depending only on events or system states with a smaller timestamp. Events corresponding to parallel communica-

tion operations in the input trace are not replayed between the parallel simulation instances. Depending on the simulation model, these communication events can be modified while preserving their event causality (as shown in §4.1) or remain unchanged (as exemplified in §4.2).

3.3.3 Output

The output of HAEC-SIM consists of the simulated output trace and a simulation log.

Output trace. The simulated output trace describes the predicted behavior of the initial application as if it were executed on the HAEC Box. It has the same file and directory structure as the input trace. Similar to the input trace, the output trace can also be visualized and analyzed with Vampir. The length (time) and size (bytes) of the output trace may differ from those in the input trace, depending on the modifications performed by the simulation modules. The number of locations corresponding of the output trace may differ depending if metrics are added or removed from the input trace. In the current design, the degree of parallelism (expressed by the number of application processes or treads) remains unchanged throughout the simulation.

Simulation log. The simulation log is the collection of all activities performed by each simulation rank during the parallel execution of HAEC-SIM. Activities comprise different pieces of information, for example a textual representation, the MPI rank of a simulation rank, or the time when the activity was performed. In addition, each activity is also assigned a verbosity level: trace, debug (default), info, warn, and fatal. Based on any of these pieces of information, the logging system can filter the activities before writing them to the standard output. The logging system can apply the activity filter based on the verbosity level at compile time. This means that the code for logging the activities which are filtered out is removed from the binary code and thus the simulation runs faster. It is recommended that the simulation log is written to a separate file in view of further offline simulation analysis.

3.3.4 Build System, Documentation, and Licensing

HAEC-SIM is built with the cross-platform, open-source build-system CMake[6]. The documentation tool doxygen[7] isn incorporated into the build process to automatically generate the user documentation of the framework. The HAEC-SIM software package and documentation is available as free software at `https://tu-dresden.de/zih/haec_sim`.

3.3.5 Testing and Verification

In order to verify the software framework, a wide range of test cases were defined. These tests are located in the directory `tests` distributed with the source code. Such test cases include *unit tests* that verify subcomponents (e.g., conversion of timestamps from nanoseconds to picoseconds, compile time filtering of activities) of the simulation framework. There are also *integration tests* that verify the framework, as a whole, and the various simulation modules. All integration tests specify a given input and output trace. The pre-simulation input traces contain a variety of communication patterns (point to point and collective). The (post-simulation) reference output trace is create once by the simulator and manually verified with Vampir. Larger traces are currently only verified in parts, by randomly selecting single events of interest (e.g., communication or computation). We are in the process of writing a tool for automatically verifying all events of interest in reference output traces. The integration tests allow to catch bugs that are introduced by changes to the simulation infrastructure or modules without having to manually verify the simulation result with each change to the software. If the theoretical simulation model is changed, the reference output trace has to be generated and verified again.

4. HAEC-SIM USE CASES

The framework is designed to be flexible and can be used in multiple ways. In this section, two use cases are described. For both cases, we use the LU application from the well-established NAS Parallel Benchmarks Suite (NPB) 3.3 [3] as a benchmark. The LU parallel benchmark solves a problem from computational fluid dynamics using the lower-upper Gauss-Seidel solver. For LU we choose the problem class C in both use cases for the following reasons: Firstly, the problem size of at least class C is needed in order to run the simulations with the desired number of MPI processes and OpenMP threads. In our case 64 in total which is suitable for the underlying topologies described below. Secondly, the total execution time and resulting input trace size is still reasonable.

4.1 Modeling Communication Performance

In the following, we demonstrate the modeling of the temporal communication performance of the LU.C benchmark on the HAEC Box. To understand the impact of the computing nodes topology on the communication performance of LU.C.64.1[8], we vary their topology. Specifically, we consider the following computing node topologies: $4 \times 4 \times 4$, $16 \times 4 \times 1$, and $64 \times 1 \times 1$. It should be noted that for these computing node topologies we assume 3D grid link topologies. In particular, the $4 \times 4 \times 4$ grid link topology represents a subset of the topology described in Section 3.1 that contains additional optical and wireless links.

We choose a low latency of 100 ns and a high bandwidth of 100 Gbit/s to simulate computing nodes connected by fast communication links. Currently the optical and wireless network links use the same characteristics. Changes to those parameters can easily be made in the configuration file. In addition, the state of the computing and communication resources is not modeled. This yields optimistic simulations with contention-free resources. As mentioned in Section 3.3.2 modeling the system state is ongoing. We use practical network coding (PNC) [20][7] as communication model. The PNC model is implemented into HAEC-SIM as the static_network_model simulation module; the implementation has been successfully verified via the usage of different test cases for which the exact solution is known. For every point-to-point (unicast) communication event read from the

[6] `http://www.cmake.org`
[7] `http://www.doxygen.org`
[8] LU.C.p.t denotes the LU benchmark of size class C running with p MPI processes and t OpenMP threads per process.

Table 1: **Simulation results for three topologies**

Computing node topology	Total running time (s)	Exclusive accum. time for appl. (s)	Exclusive accum. time for MPI (s)
$4 \times 4 \times 4$	23.096	961.729	484.939
$16 \times 4 \times 1$	23.548	961.729	513.862
$64 \times 1 \times 1$	23.509	961.729	511.344

input trace, the simulation module employs the PNC model to determine the travel time of the message (Figure 7).

The MPI processes are mapped to the computing nodes using the "xyz mapping". The communication path selection is made based on the "xyz" strategy. We obtained the input trace for the simulation by executing the instrumented benchmark LU.C.64.1 on a high performance computing system based on Intel Sandy Bridge multi-core chips with 16 cores per shared-memory node and a total of 4320 cores. The benchmark was executed on four nodes resulting in minimal node-to-node-communication. We want to answer the question: "*Which topology results in the best performance for the application under investigation?*". The simulation results are given in Table 1 and discussed below.

Total running time. The differences in the total running times are small. They stem mainly from (i) the number of hops between sending and receiving nodes, (ii) the fact that the system resources are contention-free, i.e., multiple messages can simultaneously be sent over the same link at the nominal bandwidth, and (iii) the choice of fast link parameters. For the $4 \times 4 \times 4$ topology, the maximum number of hops is 9, for the $16 \times 4 \times 1$ it is 18, and for the $64 \times 1 \times 1$ it is 63. Nevertheless, there is a noticeable difference between the fastest and the second fastest simulation run. At a first glance, it is surprising that the $64 \times 1 \times 1$ topology results in a faster execution than the $16 \times 4 \times 1$ topology. This can be explained by the choice of mapping the application processes to the computing nodes. In the case of the $16 \times 4 \times 1$ topology, the processes are mapped in such a way that the average number of hops needed for the communication is larger than in the case of the $64 \times 1 \times 1$ topology.

Exclusive accumulated time for application. The accumulated exclusive time remains constant for all three simulations as the change of topology has an impact only on the communication events, and not on the application computation functions.

Exclusive accumulated time for MPI and distribution of messages sizes. There are small differences in the time spent in MPI functions and the distribution of the message sizes, as shown in Figure 7. The different maximum number of hops between the computing nodes in the three topologies and the chosen mapping of the processes to the computing nodes are the main reasons for this behavior.

In summary, the LU.C.64.1 shows the best performance for the $4 \times 4 \times 4$ topology as the maximum number of hops is the smallest among all three topologies.

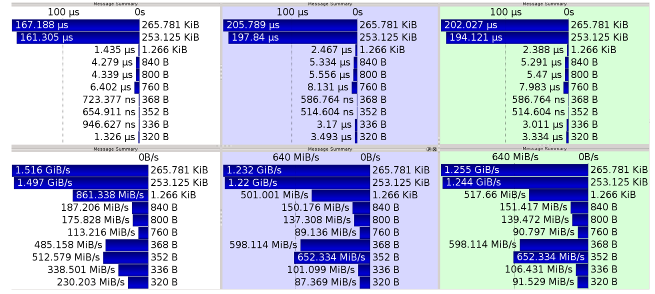

Figure 7: Message summaries (top: average message transfer time per message size, bottom: average message bandwidth per message size) for the topologies under investigation (left: $4 \times 4 \times 4$, center: $16 \times 4 \times 1$, right: $64 \times 1 \times 1$).

4.2 Computation Power Modeling

As a second use case of HAEC-SIM, we present an approach for modeling the power consumption of the HAEC Box running the LU.C.1.32 benchmark. While the HAEC development focuses on networking components, we use models of state-of-the-art CPUs to complement towards a holistic picture of the system energy consumption. The energy model for compute components is based on the work by Bircher and John [5]. As proposed in [4], the energy model is extended to account for the newer developments and prevalent techniques of compute hardware, such as multi-core, and additional processor power states. We use a dual socket Intel Sandy Bridge EP system with Xeon E5-2690 8-core / 16-thread processors. This system, called artemis, is equipped with high resolution power measurement that provides input and verification for the energy model. The power prediction uses the PAPI library to access the performance counters UOPS_DISPATCHED and OFFCORE_RESPONSE. They are recorded during execution of the LU.C.1.32 benchmark on artemis. These performance counters are recorded as synchronous metric events (i.e., illustrated as gray filled circles in Figure 2). To make sensible predictions, the energy model requires application threads to be pinned to hardware threads. This will facilitate the correlation between the core power consumption and operations within the benchmark during simulation.

For this use case, each HAEC Box computing node is considered to have the same characteristics as a single artemis processor. Therefore, threads with IDs from $\{0, 1, \ldots, 7\}$ and $\{16, 17, \ldots, 23\}$ of LU.C.1.32 are mapped to HAEC Box computing node (0,0,0) while threads with IDs $\{8, 9, \ldots, 15\}$ and $\{24, 25, \ldots, 31\}$ are mapped to node (1,0,0). Based on this mapping, the simulation module *power_estimator* instantiates two *nodes resource managers*, one for each of the two computing nodes. During the simulation, each *process location* sends all synchronous metric events (i.e., the values of the two PAPI counters for the respective core) to the resource manager responsible for the corresponding computing node. To satisfy the global causality constraint, the resource manager must process the synchronous metric events in their timestamp order, and keep track of the simultaneous metric events that originate in different locations.

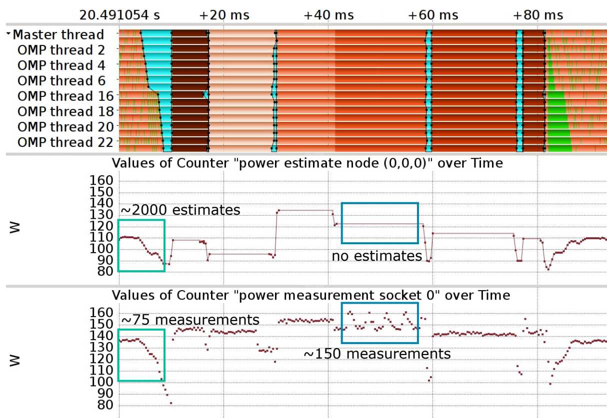

Figure 8: Results of the power estimation for LU.C.1.32 using HAEC-SIM. The top part represents the simulated event timelime of the threads mapped to the (0,0,0) computing node. The middle part represents the estimated power consumption of this node. The bottom part shows the power measurements for running the 16 threads on socket 0 of artemis. These measurements are part of the input trace.

Each synchronous metric event represents a certain system specific value that did not change since the occurrence of the previous synchronous metric event. Thus, the system state in general, and in our case its power consumption, is represented as average value for the interval between two synchronous metric events. The energy model estimates the power consumption of each computing node over the interval between two subsequent synchronous metric events. The estimation of the computing node power is done with the energy model proposed in [4].

During the processing of the synchronous metric events, the node resource manager takes all simultaneous metric events into account that correspond to the time interval of the current processed metric event. With the data of all the considered events, the resource manager can estimate the power consumption using the energy model. Each obtained power estimate and the timestamp of the corresponding synchronous metric event are stored in the node resource manager until the simulation module notifies the node resource manager that it finished processing all events. The node resource manager then sends all power estimates and their associated timestamps to the simulation module. The simulation module creates two asynchronous metrics, i.e. "power estimate node (0,0,0)", and "power estimate node (1,0,0)". The power estimates are then written by the simulation module to the corresponding new metric location as asynchronous metric events.

A snippet of the output trace for the LU.C.1.32 benchmark is shown in Figure 8. In contrast to the power measurements (bottom part in Figure 8), the resolution of the power estimation (middle part in Figure 8) is variable. While the measurements are taken at fixed intervals, the amount of time between two estimates depends by design on the time interval between the two synchronous metric events corresponding to the performance counters. Thus, even though the estimates are written as asynchronous metric events, their data points are aligned to the synchronous metric events. This fact can be seen as an advantage, e.g., the resolution of the estimation until the first barrier at +10 ms is higher than the resolution of the measurement. It can also be a disadvantage, e.g., the estimation between +40 ms and +60 ms gives only a few data points. In this case the variability shown by the measurement is not revealed by the estimates. Further this makes it more easy to attribute energy values to specific function intervals, which can be challenging with measurements that are not aligned to function calls. It is also possible to read the performance counters asynchronously with PAPI by using an metric plugin extension of Score-P. By doing so the performance counter values are available at regular time intervals also resulting in power estimates at regular time intervals similar to measurements.

The proposed power estimation approach of application computations represents a proof of concept that can be used for more advanced energy modeling. The model scales with the number of nodes in the simulated system, provided that the input trace contains a sufficient number of threads to saturate the nodes of the simulated system. The power estimation depends on the temporal granularity of the synchronous metrics in the input trace. The current approach can be extended to include other types of application activities (e.g., computation, communication, and combinations thereof).

5. CONCLUSIONS AND FUTURE WORK

In this paper we motivate the need for exploring the design alternatives of a highly adaptive energy efficient computing platform and its applications. This requires an end to end simulation framework supporting the integration of abstract models describing the platform under design and its applications. The framework takes as input execution traces of applications on an existing system, applies the simulation models to the input trace, and generates an output trace representing the simulated behavior of applications on a future target platform. We demonstrate the proposed framework two simulation scenarios that are applied to a parallel application. In particular communication models allow to evaluate the impact design decisions such as the topology on application execution. Energy models that are based on measurements of state of the art hardware, enhance the simulation to put a focus on energy optimiztion rather than only considering performance.

There are several ongoing and future work directions. Ongoing efforts include: The development of mapping strategies that consider the communication patterns of the application is an important ongoing work. Further we already implemented a model for unicast communication in the presence of errors/attacks that we aim to verify. The energy energy models are going to be extended for communication operations. A central aspect is also the modeling of shared system resources to address contention. Future work directions include: modeling of multicast communication; development of support for migration of tasks across computing nodes to increase performance or decrease energy costs; development of a hybrid communication model that supports dynamic latency, bandwidth, and path selection.

6. ACKNOWLEDGMENTS

This work is supported by the German Research Foundation (DFG) in the Collaborative Research Center 912 “Highly Adaptive Energy-Efficient Computing”. The authors would like to thank Andreas Knüpfer and Joseph Schuchart of Technische Universität Dresden for their earlier contributions leading to the present simulation framework. Acknowledgements also go to Elke Franz and Stefan Pfennig of Technische Universität Dresden for their work on the implementation of the network models in the simulation framework.

7. REFERENCES

[1] E. Argollo, A. Falcón, P. Faraboschi, M. Monchiero, and D. Ortega. COTSon: Infrastructure for full system simulation. *SIGOPS Op. Sys. Review*, 43(1), 2009.

[2] R. Aversa, B. Di Martino, M. Rak, S. Venticinque, and U. Villano. Performance Prediction Through Simulation of a Hybrid MPI/OpenMP Application. *Parallel Computing*, 31(10-12):1013–1033, Oct. 2005.

[3] D. Bailey, E. Barszcz, J. Barton, D. Browning, R. Carter, L. Dagum, R. Fatoohi, S. Fineberg, P. Frederickson, T. Lasinski, R. Schreiber, H. Simon, V. Venkatakrishnan, and S. Weeratunga. The NAS parallel benchmarks. RNR Technical Report RNR-94-007, NASA, March 1994.

[4] M. Bielert. Evaluating power estimation techniques: A methodological approach. Master's thesis, Technische Universität Dresden, 2015.

[5] W. Bircher and L. John. Complete System Power Estimation Using Processor Performance Events. *Computers, IEEE Transactions on*, 61(4):563–577, April 2012.

[6] S. Böhm and C. Engelmann. xSim: The extreme-scale simulator. In *Proc. of the Intl. Conf. on High Perf. Comp. and Sim. (HPCS)*, pages 280–286, Istanbul, Turkey, July 4-8, 2011. IEEE Computer Society, Los Alamitos, CA, USA. Acceptance rate 28.1% (48/171).

[7] F. M. Ciorba, T. Ilsche, E. Franz, S. Pfennig, C. Scheunert, U. Markwardt, J. Schuchart, D. Hackenberg, R. Schöne, A. Knüpfer, W. E. Nagel, E. A. Jorswieck, and M. S. Müller. Analysis of parallel applications on a high performance–low energy computer. In *Proc. of the Euro-Par 2014 Workshops: 7th Workshop on UnConventional High Performance Computing (UCHPC)*, volume 8806 of *Lecture Notes in Computer Science*, pages 474–485. Springer, Dec 2014.

[8] W. E. Denzel, J. Li, P. Walker, and Y. Jin. A Framework for End-to-End Simulation of High-performance Computing Systems. *SIMULATION*, 86(5-6):331–350, May 2010.

[9] G. Fettweis, W. Nagel, and W. Lehner. Pathways to servers of the future. In *Proc. of the Design, Automation Test in Europe Conference Exhibition (DATE)*, pages 1161–1166, Mar 2012.

[10] R. M. Fujimoto. Parallel and Distributed Simulation. In *Proc. of the Winter Conf. on Sim.*, pages 122–131. IEEE, 1999.

[11] M.-A. Hermanns, M. Geimer, F. Wolf, and B. J. N. Wylie. Verifying Causality between Distant Performance Phenomena in Large-Scale MPI Applications. In *Proc. of the Euromicro Intl. Conf. on Par., Dist. and Network-based Proc.*, pages 78–84, Feb. 2009.

[12] F. Hillier and G. Lieberman. *Introduction to Operations Research*. McGraw-Hill Higher Education, 9th edition, 2010.

[13] T. Hoefler, T. Schneider, and A. Lumsdaine. LogGOPSim: Simulating Large-scale Applications in the LogGOPS Model. In *Proc. of the 19th ACM Intl. Symp. on High Perf. Dist. Comp.* ACM, June 2010.

[14] A. Knüpfer, H. Brunst, J. Doleschal, M. Jurenz, M. Lieber, H. Mickler, M. S. Müller, and W. E. Nagel. The Vampir performance analysis tool-set. In M. Resch, R. Keller, V. Himmler, B. Krammer, and A. Schulz, editors, *Tools for High Performance Computing*, pages 139–155. Springer, Jul 2008.

[15] A. Knüpfer, R. Dietrich, J. Doleschal, M. Geimer, M.-A. Hermanns, C. Rössel, R. Tschüter, B. Wesarg, and F. Wolf. Generic support for remote memory access operations in Score-P and OTF2. In A. Cheptsov, S. Brinkmann, J. Gracia, M. M. Resch, and W. E. Nagel, editors, *Tools for High Performance Computing 2012*, pages 57–74. Springer Berlin Heidelberg, 2013.

[16] A. Knüpfer, C. Rössel, D. Mey, S. Biersdorff, K. Diethelm, D. Eschweiler, M. Geimer, M. Gerndt, D. Lorenz, A. Malony, W. E. Nagel, Y. Oleynik, P. Philippen, P. Saviankou, D. Schmidl, S. Shende, R. Tschüter, M. Wagner, B. Wesarg, and F. Wolf. Score-P: A joint performance measurement run-time infrastructure for Periscope, Scalasca, TAU, and Vampir. In H. Brunst, M. S. Müller, W. E. Nagel, and M. M. Resch, editors, *Tools for High Performance Computing 2011*, pages 79–91. Springer Berlin Heidelberg, 2012.

[17] J. Labarta, S. Girona, and T. Cortes. Analyzing scheduling policies using Dimemas. *Par. Co.*, 23(1-2):23–34, Apr. 1997.

[18] L. Lamport. Time, clocks, and the ordering of events in a distributed system. *Commun. ACM*, 21(7):558–565, July 1978.

[19] A. Marowka. Back to thin-core massively parallel processors. *Computer*, 44(12):49–54, 2011.

[20] S. Pfennig, E. Franz, F. M. Ciorba, T. Ilsche, and W. E. Nagel. Modeling communication delays for network coding and routing for error-prone transmission. In *Proc. of the 3rd Intl. Conf. on Future Gen. Comm. Techn.*, pages 19–24. IEEE, Aug 2014.

[21] S. W. Sherman and J. C. Browne. Trace driven modeling: Review and overview. In *Proc. of the 1st Symp. on Simul. of Computer Sys.*, pages 200–207. IEEE Press, 1973.

[22] M. Tikir, M. Laurenzano, L. Carrington, and A. Snavely. PSiNS: An Open Source Event Tracer and Execution Simulator for MPI Applications. In *Proc. of the 15th Intl. Euro-Par Conf. on Par. Proc.*, page 148, 2009.

[23] E. Totoni, A. Bhatele, E. J. Bohm, N. Jain, C. L. Mendes, R. M. Mokos, G. Zheng, and L. V. Kale. Simulation-based performance analysis and tuning for a two-level directly connected system. In *Proc. of the 17th IEEE Intl. Conf. on Par. and Dist. Sys.*, pages 340–347, 2011.