Visualization of Simulation Results for the PERCS Hub Chip Performance Verification

Andreas Doering
IBM Research - Zurich
Säumerstrasse 4
8803 Rüschlikon, Switzerland
ado@zurich.ibm.com

Hanspeter Ineichen
emerge engineering
Brambergstrasse 11
Lucerne, Switzerland
hpi@emen.ch

ABSTRACT

Performance verification ensures that an implementation of a given architecture will deliver the expected performance. The Productive, Easy-to-use, Reliable Computing System has particularly high performance goals measured at the progress of technology. Its Hub chip constitutes the main network and I/O component, and therefore strongly affects the system performance. Performance verification requires the rapid detection of performance deficits in tests with regular request patterns as well as the analysis of sophisticated problems in more complex test situations. In this paper, visualization methods and tools used in the performance verification of the PERCS Hub chip are presented. Not only existing tools, such as spreadsheets, were integrated, but also a new dedicated tool was developed.

Categories and Subject Descriptors

C.4 [**Performance of Systems**]: Modeling techniques; I.6.8 [**Simulation and Modeling**]: Type of Simulation—*Discrete event*

1. INTRODUCTION

PERCS (Productive, Easy-to-use, Reliable Computing System) is a supercomputer system developed and produced by IBM*, partially supported by the Defense Advanced Research Projects Agency (DARPA). The funding contract sets formal performance targets for a set of benchmark applications. The initial funding supported the architecture and technology development, which in return projected the expected performance using among other things high-level simulations. Therefore, during development of the two major chips, the POWER7* processor and the Hub chip, several detailed high-level simulation models acted as blueprints for the implementation.

Even though most chip developments today contain some degree of performance verification, the effort for this aspect in PERCS was much higher than usual, because of the funding terms and newly introduced technology. On one hand, the new technology makes the chips more expensive, which should thus be accompanied with a corresponding performance increase, on the other hand, the lack of experience with newly introduced architecture aspects bears a greater risk of incurring flaws that impact performance. Performance verification should guarantee that all performance targets are met, and, if not, it should provide insight why performance is insufficient. The first task requires the definition and maintenance of an appropriate set of test cases and their performance characteristics, whereas the second task involves the development of analysis methods for the case at hand.

Performance analysis of processor cores and memory controllers is a well-established discipline. Because the high-level models of processors are typically cycle accurate - at least for instruction issue and completion, the performance verification is a cycle-by-cycle comparison. Such an approach is not possible for network components where entire networks for extended simulation times have to be simulated, representing a typical application run. Therefore, the performance verification is the comparison of similar behavior. To our knowledge this has not been done before. In particular, we are not aware of visualization methods for the comparison of network traffic behavior in two different models.

An important method for the analysis of performance deficiencies is the visualization of the details of a simulation run. As varied as the performance problems are as wide the visualization schemes need to be. Therefore, the development of a new visualization scheme needs to be fast, but the resulting graphs need to be intuitive and easy to correlate with the simulation details. As gate level simulations as part of the verification process are time consuming, the analysis needs to be organized as a post-processing step. Otherwise, a change in the visualization would require the repetition of the simulation run.

This paper presents several of the visualization methods that were used and we demonstrate how the resulting graphics allow the identification of the cause of a performance problem. This paper will not cover all performance test

*IBM and POWER7 are trademarks of International Business Machines Corporation, registered in many jurisdictions worldwide. Microsoft and Excel are a trademarks of Microsoft Corporation in the United States, other countries, or both. UNIX is a registered trademark of The Open Group in the United States and other countries. Other product and service names might be trademarks of IBM or other companies.

SIMUTOOLS 2011, March 21-25, Barcelona, Spain

DOI 10.4108/icst.simutools.2011.245536

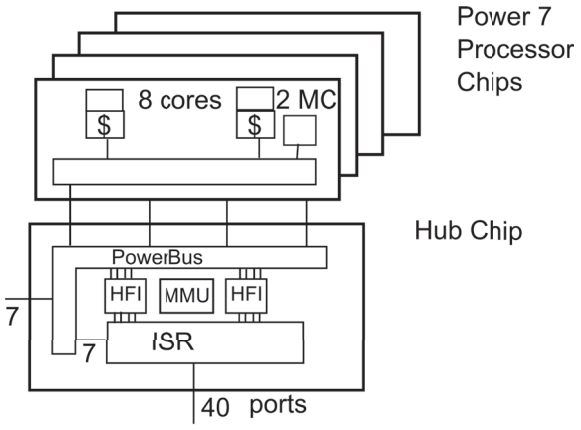

Figure 1: Node of a PERCS system consisting of the Hub chip and four processor chips

cases, only a few that required a visualization effort. In this paper examples are chosen from simulations during development, that show performance deficits. These deficits have been identified and fixed in the final design, partially by the help of the presented visualization methods. It should be clear that the final chip does not exhibit these weaknesses. However, only by selecting simulation runs that show problems, the value of the visualization methods can be demonstrated. It should be noted that the visualization tools were implemented as needed and their refinement was only done as far as necessary as the time pressure and work load of the project itself was high.

The paper is structured as follows: The simulated Hub chip is introduced in Section 2. Section 3 covers the simulation environment used for high-level reference simulations and for verification runs of the implementation. Sections 4, 5, and 6 describe three visualization methods with increasing effort.

2. THE PERCS HUB CHIP

The PERCS Hub chip [Arimilli et al.(2010)] is the main component of the PERCS interconnection network that allows up to 65536 POWER7 processor chips to be connected (Figure 1). Each processor chip contains 8 processor cores. Such a large-scale interconnect requires an efficient network interface with user memory access. The PERCS Hub chip contains two network interfaces called HFI (Host Fabric Interface). Furthermore it contains a low-latency, high-throughput 55-port switch (Integrated Switch Router, ISR), a Memory Management Unit, and other units, that are not relevant for this paper. The connection to the POWER7 is provided by the split-transaction, high-performance, hierarchically cache-coherent PowerBus.

The HFI participates in the PowerBus protocol in the same way a processor does: to access memory (for reading send data or writing receive data), it creates a snoop request, awaits the snoop response, and handles data accordingly. This allows receive data to be injected into the processor caches or a copy to be fetched from cache on a read. The snoop response can also demand a retry of the request if the target unit is overloaded or an address conflict for atomic operations is detected. Each HFI has 4 data ramps to the switch and to the bus in both directions. Only in this way can the high throughput be achieved without increasing the data granularity and maintaining a realistic clock frequency. On the send side, the origin of the message determines the data ramp, on the receive side this is done by the scheduler in the ISR.

3. SIMULATION ENVIRONMENT

The logic aspect of chip development is done using VHDL (Very High Speed Integrated Circuit Hardware Description Language), which is synthesized to a netlist, that in turn is further processed for the final implementation. The VHDL description of the chip is also used in simulation to verify correct behavior. The VHDL source code is translated into an intermediate format called a "model", which is used by the simulator MESA together with a model-specific software module "Run Time Extension". This rtx contains drivers and monitors that can assign values to input signals of the model and test the value of any signal in the model. It also controls the execution of the simulation itself, for instance with respect to clocks. The ideas behind this structure are described in [Wile et al.(2005)]. The development of the rtx for a chip project entails a considerable effort. Therefore, performance verification uses the same code base, extending it by additional stimuli generation and monitoring capabilities. In this way the effort is reduced and functional errors can be detected during performance test-case runs. The monitors in the rtx are software components that observe a particular interface and check whether it behaves correctly. Furthermore, the monitors can print each transaction into a common trace file (see Figure 2 for an example).

The line in Figure 2 corresponds to a command request of a given address and type (cache inhibited partial write) from the interface of a processor to the PowerBus. It belongs to a write operation to the so-called "doorbell register" of the HFI. This will in turn activate the HFI to read the data for a network packet. If a test case is to analyze the send latency for a packet, this event would represent the start time. However, to find the corresponding packet at the network side of the HFI, the corresponding read data request has to be found. This will use a different address and type, and the connection between the two is found in a configuration section of the trace file for each doorbell register. When the read request has been found, the read transaction needs to be followed to find the data transfer, which can then later be matched with the send data. All these steps can turn out to be complicated, for instance if the read address is first translated or the send data rotated when the packet start in memory is not aligned with a cache line boundary. Some of these steps are already provided by the rtx code for functional verification.

For simple performance test cases, input generation can be done using the code for random input generation, which is part of the rtx for functional verification. For more complicated tests, e.g. driving a bus arbiter with a moving hotspot, or permutations, either specific code or the use of stimuli input files is necessary. In particular, to test the performance of the ISR and an entire node, trace files from high-level simulation models were used.

There are two high-level network models used for the PERCS system performance prediction, one OMNET-based, event-driven, which can scale to the entire PERCS network size of 16K Hubchips [Denzel et al.(2008)] and a second,

```
Cyc 0034939:  CmdReq ; seq=385067 pbid=4 ttag=0x01080:n0.c2.EX1.NC.st.0x0.tid0 addr=1267:4c180110
ttype=0x1b80(x37_x08):N ack:ci_pr_w bytes=8 snpport=0 hubpump=1
```

Figure 2: Example trace file line

queue-based. From both models, traces relating to relevant benchmarks for selected nodes were extracted.

Main performance figures of interest are the latency and the throughput of various units and their operations, for instance, snoop transactions on the PowerBus and packet receive in the HFI. Even though both terms are intuitive, a precise definition is needed: for instance, many events such as receiving a cache line at the bus interface, consume a few clock cycles. What are the start and end times to determine latency? There were two principles for the two figures: latency should be additive when a transaction passes through several units sequentially, and throughput should be consistent when measured over consecutive time intervals. Therefore, for both terms, the same point in time of the event, such as the rising edge of the first data beat, is used. This can increase the complexity of the trace-file analysis.

4. SPREADSHEETS

Spreadsheet programs, such as Microsoft Excel* software or OpenOffice CALC offer the generation of diagrams in various styles and allow the import of data as text files. Using a combination of an analysis script and a spreadsheet, quite complex situations can be illustrated. Because of the simplicity of the approach, it can be used even for confirmation of correct behavior or to get an impression of the behavior of a hardware unit before a test case is created.

As an example, consider a system with one type of transaction, where each transaction consists of two events, e.g., the submission of a data item into a bus and its reception at a target unit of the bus. A simple best-case test with maximum-rate injection can be visualized by using two columns in a spreadsheet, which contain the time of the events: one for each event type. The content of the spreadsheet can typically be generated very easily: The `grep` utility can be used to extract the lines for a certain event type from the simulation trace file. Tools such as Unix* `cut` will extract the portion of each line corresponding to the simulation cycle. Finally, the files with the cycles for both event types can be combined using `pr`. An example of a command trace generated this way is shown in Figure 3. Each line corresponds to one bus transaction: the vertical axes represents simulation time, the horizontal axes corresponds to the event, 1 for the start, and 2 for the end of a point-to-point data transfer between one pair of bus units.

It can be seen that the transactions get slower over time. This is a typical behavior when the system contains a certain amount of buffers and a limited processing or transmit capacity: when the buffers get filled, backpressure sets in and reduces the injection rate, and the latency becomes higher due to buffering delays.

One pitfall with the spreadsheet approach is the handling of empty entries. If transactions on several interfaces are combined into one spreadsheet it can happen that one interface is finished before the others, and the columns in the spreadsheet get mixed in the wrong way. This can be avoided by fixed-width columns, but requires more effort in the preparation scripts.

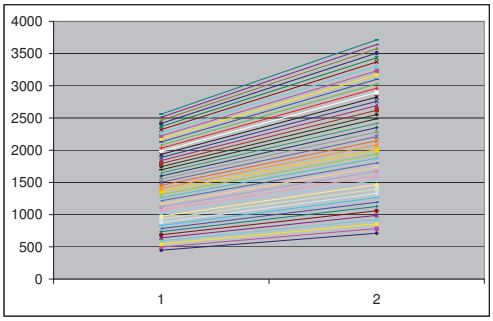

Figure 3: Using a spreadsheet to illustrate one transaction stream

5. DISTRIBUTION OF TAGS

One of the most complex situation is the analysis of the Distributed Memory Access Benchmark [Aggarwal et al.(2009)]. This test case generates randomly distributed read-modify-write operations for the memory at any processor in the system. The operations (operation type, virtual memory address, operand data word) are transported through the network and issued by the Hub chip as a bus transaction to the target memory controller. The target performance of this benchmark is one of the most aggressive goals of the PERCS system. The performance of this benchmark depends on many components, such as the processor core that generate the operations, the network interface for sending and receiving the operations, the network to exchange them, the MMU to translate the virtual addresses into physical ones, the system bus to carry the target operations and the memory controller. In particular the interaction between them had to be covered. Therefore, simulations covering many of these components were crucial to verify the implementation.

A model consisting of four processor chips (without processor cores, but including the caches and the memory controller), and one Hub chip was used. The operations are injected into the model at the core-to-nest interface by write transactions which are read from a text file. Most of the write operations create the send packets in the caches, and few operations write to control registers in the network interface. The network interface in the model will read the send data (found in the caches) and will send the packet out. A part of the set of packets is returned in the switch directly to one of the network interfaces. The remaining packets will leave the model through the links and are captured by the rtx code. This code will modify the destination address of the packets and re-inject them into the same link after a predefined delay. The packets received by the network interface are turned into read-modify-write transactions on the

system bus, which are eventually processed by the memory controller. The memory itself is emulated by driver code which will check that the right operations and data were used.

This simulation takes approximately 24 hours for 4000 processed operations. For this reason, each simulation run needs to be analyzed in detail, deriving a maximum of insight from it. The first simulation runs showed a low performance at the memory controllers even though the maximum number of concurrent operations for each memory controller had not been reached. This could be shown using a spreadsheet based visualization of the arriving vs. completed operations at each memory controller. The cause was easily identified: because the target address range for the read-modify-write operations was chosen too small, two operations frequently addressed the same cache line. As the operations are considered atomic, the operations had to be canceled and scheduled for retry.

Even after fixing this, the performance did not meet the target, but this time the cause was not as easy to identify. There are too many resources and arbitrations involved, and after checking them individually, insight in the overall behavior was needed. To understand the problem, a complex visualization was created. The following aspects needed to be visible:

- Command requests, snoop responses, and data movement with their timing,
- correspondence between them,
- addressed Memory controller, and
- use of request tags

One suspicion was that a "moving hot spot" at the memory controllers or the data links between was causing the performance limitation, i.e., that one or few of the memory controllers were overloaded for a short duration, although in the long term all have the same load. By using different colors based on the destination memory controller and combining all events of one transaction into a geometric shape, most of the requirements listed could be met. The time of an event is represented by its horizontal position. However, the use of the request tags is more difficult as the number of available tags is quite high and it is difficult to identify the exact point when the use of a tag ends. A numerical representation of the tags is nearly meaningless and should not be used directly in visualization. Therefore, for the vertical position, an allocation/deallocation scheme of rows was used, resulting in a dense diagram. As rows are reused bottom-up, the height of the diagram relates to the number of tags used at any time. Hence, one aspect, i.e. , an underuse of tags, or a tag leak would be immediately visible. Figure 4 shows an excerpt of such a diagram. Even when generating the diagram only for a short time interval from a trace run, it would be far too large to include it in the paper. One transaction corresponds to a horizontal line starting with a right arrow, and ending with an oval, which in most cases is not visible because it is overlaid by a left arrow corresponding to the positive snoop response. Each additional right arrow corresponds to a retried request. The overlapping rules out another set of potential causes for performance problems: data movement. As data movement starts at the earliest possible moment in nearly all cases, there is obviously no

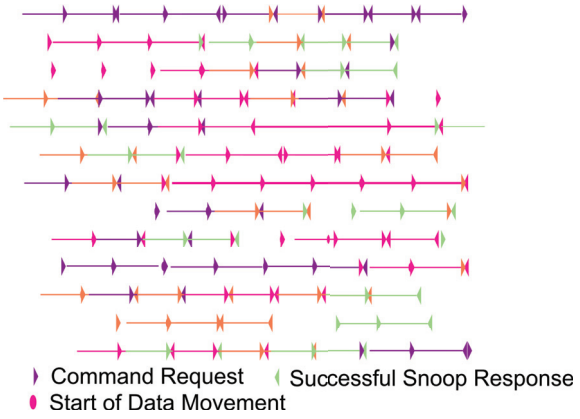

Figure 4: Example illustration of tag usage over time

data back pressure. Unfortunately, another insight will not be visible very well if the paper is printed in black and white: there is no concentration of one color at any part of the diagram. This rules out the suspected moving hot spot. Still, many transactions require several retries. It is also seen that the HFI is able to issue commands, including retries, at a high rate. Otherwise there would be longer lines without retry arrows for a significant number of transactions, a fact that can also be seen with a short glance. The visualization was implemented using a perl script that generates a Scalable Vector Graphics file. This can be viewed with the SVG drawing program `inkscape`. SVG enables the use of labels for graphical elements and this feature was used to mark all events. By generating the labels on the base of identifiers in the trace file, a feedback from the illustration to the simulation details is provided.

6. TOOL FOR CROSSBAR SWITCH

The 55-port crossbar switch of the PERCS hub chip requires a particularly large set of test cases, covering latency and throughput of every input-output pair, the different routing modes, and the behavior for traces derived from high-level simulations. In contrast to the previously described aspects, the presentation of the results needs to cover several simulation runs. Therefore, a spreadsheet or script-based approach is impractical, and a C++-based tool using the board library [Fourey(2007)] was developed. To get an overview, a table in HTML format is created by the tool. HTML allows the fields of the table to be highlighted , which is used to mark runs with sufficient performance in green; otherwise the fields are marked in yellow to red depending on the degree of performance deficiency. This coloring provides a quick impression of the situation with over 3000 individual performance figures, i.e., whether the failed targets were few individual spots, dominate or have some regularity.

Furthermore, simulations driven were run by traces extracted from high-level models as mentioned before. Such a stimuli trace contains a long list of flits (FLow control unITs, 128-byte data blocks transmitted over the links and buffered in the switches), together with the time they enter and leave the switch or the network interface. Furthermore, the time, virtual channel [Duato et al.(2002)], and number of credits

returned are also recorded. When the VHDL-model is driven with such a high-level trace, a flit is injected at its specified port (network link or HFI) as soon as a credit is available after its time stamp. For credit return, also a combination of trace information and simulation run is used: it is clear that a credit cannot be returned before the link round-trip time after the sending of a flit. However, in a network congestion situation, credits are returned much later.

The comparison of the simulation run with the VHDL-model to the high-level trace it was driven from, is a challenge. There are cases where the scheduling decisions for the next packets to be sent over a link differ. This means that the latency through the unit will change for two flits, and in turn influence the input-throughput for two links. Such a subtle difference can impact a longer tail of flits. This is an expected deviation of the high-level model from the implementation. The question is in which cases do such differences cause performance differences on the larger scale. As a first approximation, one can use histograms or sortograms of the positive and negative differences of the flit sending times. One can further automate the test whether a flit that leaves on a link later than in the stimuli trace could not have been transmitted earlier, i.e., the test that the link did always send a flit whenever a credit was available after the arrival of the flit considered (plus some time for internal processing). Such an approach would cover only a small set of possible performance problems, at the cost of a significant programming effort. Therefore, various visualization methods were used instead to quickly identify problematic sections (in time or set of ports) that could then be investigated in more detail.

The first type of diagram uses the time in the high-level trace on the x-axes and the time in the VHDL-level simulation on the y-axes. Each flit is represented as a vector. The starting point of the vector corresponds to the injection of the flit into the switch and the end of the vector is derived from the time when the flit leaves the switch, in both the high-level simulation and in the simulation of the implementation model. In the ideal case, where the timing of the both models is identical, all vectors would be located on the diagonal of the diagram. The starting point of the vectors cannot lie under the diagonal, but the end points can, indicating that the implementation works faster than the high-level model assumed. If the implementation is systematically slower than the high-level model assumed, most of the vectors would be located above the diagonal. If required, also the credits could have been added, but this turned out to be unnecessary. However, more information can be added by using color, for instance to distinguish incoming or outgoing port, the routing type, or flit type. Only the header flits require a routing table lookup, and because of the network topology, not all header flits do. Figure 5 contains all trace-driven traffic sent from the HFI into the ISR. As can be seen, some interfaces are much slower than the trace file. The problem was identified on this basis as an unfortunate correlation between requests in the crossbar switch arbiter.

An overall picture with traffic from all ports from a different simulation run is shown in Figure 6. It can be seen that the performance of the implementation starts to fall behind that of the reference trace at a certain point in time. This simulation uses the stimuli trace from a second high-level model (CSim Queing-based model). For technical reasons, no transfer time is used, only the arrival times and destination ports. Therefore, the vectors are all vertical. It can be immediately seen at which point in the simulation the design behavior deviates from the reference, so that more detailed investigation can focus on this point.

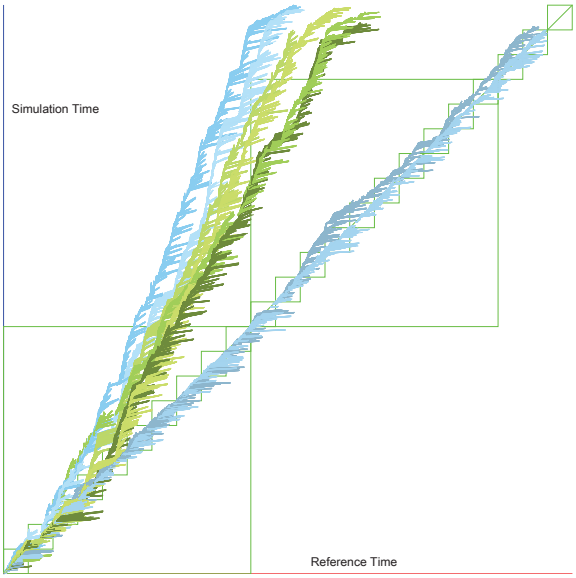

Figure 5: HFI ramp usage for send traffic

Both figures represent 2.3 ms of simulated time in the real system. The simulation run of the VHDL model requires around 24 hours. There are of course methods to speed-up the VHDL-simulations for performance analysis purposes, e.g. removal of error checking and error correcting circuits. However, the more productive approach is simulating more short cases of critical behavior.

7. CONCLUSION

Many high-level simulation environments support visualization. Low-level simulations typically provide only waveform viewing and otherwise use textual reporting. From the experience reported here this is sufficient. The extraction of performance-related information from a textual trace file and the generation of illustrations covering the interesting aspects in post-processing can be done ad-hoc or by a more elaborate approach as described in this paper. Note that neither of the more detailed aspects, i.e., tag usage or ramp distribution, was anticipated to become relevant when the project started. It is our experience from the PERCS Hub chip and other projects that performance aspects can be quite complex and difficult to predict. Even a good high-level simulation cannot predict the impact of small functional deviations in the implementation. Today's deep-submicron technologies present high challenges to the implementation of a given architecture, including power, wire-length, area, and reliability issues. A universal tool that could cover the wide scope of aspects as demonstrated by the examples would have to be quite sophisticated, and it is questionable whether the productivity including learning to use such a tool would be higher than plain programming. Currently, work on other networking chips including high-level simulation is ongoing. We are discussing whether

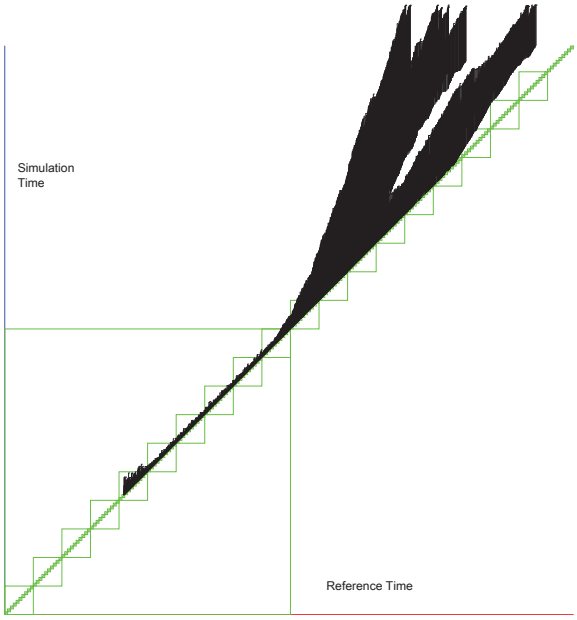

Figure 6: Simulation driven by second high-level model

performance verification during the planned implementation of these chips should be carried out and whether some of the presented visualization tools should be extended for the task.

Acknowledgement

This material is based upon work supported by the Defense Advanced Research Projects Agency under its Agreement No. HR0011-07-9-0002. This work is also supported by the Blue Waters sustained-petascale computing project, which is supported by the National Science Foundation (award number OCI 07-25070) and the state of Illinois. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the funding agencies. Hanspeter Ineichen was employed by Supercomputing Systems while carrying out work described in this paper at the IBM Research – Zurich Laboratory.

7.1 References

8. REFERENCES

[Aggarwal et al.(2009)] V. Aggarwal, Y. Sabharwal, R. Garg, and P. Heidelberger. HPCC RandomAccess benchmark for next generation supercomputers. In *IPDPS '09: Proceedings of the 2009 IEEE International Symposium on Parallel & Distributed Processing*, pages 1–11, Washington, DC, USA, 2009. IEEE Computer Society. ISBN 978-1-4244-3751-1.

[Arimilli et al.(2010)] B. Arimilli, R. Arimilli, V. Chung, S. Clark, W. Denzel, B. Drerup, T. Hoefler, J. Joyner, J. Lewis, J. Li, N. Ni, and R. Rajamony. The PERCS high-performance interconnect. In *Proceedings of 18th Symposium on High-Performance Interconnects (Hot Interconnetcs 2010)*. IEEE, Aug. 2010.

[Denzel et al.(2008)] W. E. Denzel, J. Li, P. Walker, and Y. Jin. A framework for end-to-end simulation of high-performance computing systems. In *Simutools '08: Proceedings of the 1st International Conference on Simulation Tools and Techniques for Communications, Networks and Systems & Workshops*, pages 1–10, ICST, Brussels, Belgium, Belgium, 2008. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering). ISBN 978-963-9799-20-2.

[Fourey(2007)] S. Fourey. LibBoard - a C++ library for simple Postscript, SVG, and XFig drawings. http://libboard.sourceforge.net, 2007.

[Wile et al.(2005)] B. Wile, J. Goss, and W. Roesner. *Comprehensive Functional Verification: The Complete Industry Cycle (Systems on Silicon)*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2005. ISBN 0127518037.

[Duato et al.(2002)] J. Duato, S. Yalamanchili, and L. Ni *Interconnection Networks: An Engineering Approach*. Morgan Kaufmann Publishers Inc., San Francisco, USA, 2002. ISBN 1558608524.