Open access • Proceedings Article • DOI:10.1109/IPDPS.2011.157

Distributed Resource Reservation in Massively Parallel Processor Arrays

— Source link

Vahid Lari, Frank Hannig, Jürgen Teich

Institutions: University of Erlangen-Nuremberg

Published on: 16 May 2011 - IEEE International Symposium on Parallel & Distributed Processing, Workshops and Phd Forum

Topics: Massively parallel, Parallel processing (DSP implementation), Degree of parallelism and Scalability

2011 IEEE International Parallel & Distributed Processing Symposium

Distributed Resource Reservation in Massively Parallel Processor Arrays

Vahid Lari, Frank Hannig, and Jürgen Teich
Hardware/Software Co-Design, Department of Computer Science
University of Erlangen-Nuremberg, Germany
Email: {vahid.lari, hannig, teich}@cs.fau.de

***Abstract*—This paper proposes a methodology for applications to automatically claim linear arrays of processing elements within massively parallel processor arrays at run-time depending on the available degree of parallelism or dynamic computing requirements. Using this methodology, parallel programs running on individual processing elements gain the capability of autonomously managing the available processing resources in their neighborhood. We present different protocols and architectural support for gathering and transporting the result of a resource exploration for informing a configuration loader about the number and location of the claimed resources. Timing and data overhead cost of four different approaches are mathematically evaluated. In order to verify and compare these decentralized algorithms, a simulation platform has been developed to compare the data overhead and scalability of each approach for different sizes of processor arrays.**

I. Introduction

Growing demands for high computational capabilities for computer systems lead to employ more hardware and software parallelism by increasing the number of processing elements placed in a multiprocessor system-on-a-chip (MPSoC). Because of this technological shift from multi-core to many-core system-on-a-chip designs, controlling and programming of 100s to 1000s of processing elements are becoming one of the major challenges for system designers. In addition, huge diversity in the application domain signals to have more flexible architectures that can be reconfigured according to the application requirements. Whereas for a single application, the optimal mapping onto an array of processors may be computed at compile-time, which holds in particular for loop-level parallelism and the corresponding programs, a static mapping might not be feasible for the execution at run-time because of time-variant resource constraints or because the degree of exploitable parallelism may be data-dependent, or because of temporal and permanent failures. Thus, in the future, applications and architectures should exploit dynamic resource requirements while avoiding fully centralized and not scalable control of their execution. The main idea of our methodology is to give processors the ability to explore neighbor processors and to distribute and execute in parallel their application. This paper investigates different techniques for organizing, gathering and transmitting information about the number and the location of available resources. The proposed approaches are evaluated using mathematical models and their functionality is compared in different experimental scenarios. The rest of this paper is organized as follows: A brief overview of related work is given first. In Section III, our distributed resource reservation methodology is explained shortly. Section IV proposes different concepts and algorithms that can be employed in such systems to return back the result of resource explorations. A mathematical evaluation of the proposed methods, in terms of timing and hardware cost, is given in Section V. A simulation environment is presented in Section VI to compare the proposed approaches. Finally, the paper is concluded in Section VII.

II. Related Work

In this section, we give an overview of the recent projects from academia on reconfigurable multi-core architectures. In the TRIPS project [1], an array of small processors is used for the flexible allocation of resources dynamically to different types of concurrency. In the CAPSULE project [2], the authors describe a component-based programming paradigm combined with hardware support for processors with simultaneous multi-threading (SMT) in order to handle the parallelism in irregular programs. Here, an application is being dynamically parallelled at run-time. Resano and others [3] developed a hybrid design/run-time pre-fetch heuristic that schedules reconfigurations at run-time but carries out the scheduling computations at design-time. The algorithm mapping for PACT XPP [4] is done in a similar way. All the aforementioned approaches have in common that if the number of available resources should change, the entire application has to be recompiled and mapped again. Furthermore, the above architectures are controlled centrally and often provide no mechanisms to monitor the utilization of the computing resources. In order to tackle these problems, we introduce a distributed hardware-based approach to monitor the system utilization and automatically adapt the mapped applications according to the amount of available resources at run-time.

III. Distributed Resource Reservation

In the seminal paper by Teich et al. [5], a novel paradigm of adaptive computing called **Invasive Computing** is introduced. The main idea behind *invasion* is to give parallel programs the capability of requesting and temporarily claiming processing, communication, and memory resources in their neighborhood, to then execute in parallel the given program using the claimed resources, and the capability of subsequently freeing these resources again. At the beginning, a program (mapped to a PE that we call it *master PE*) may define its resource requirements which leads to call for an exploration in order to capture available resources and reserve them for subsequent parallel execution. After that, an infection is performed so to copy and then execute the application code to the captured

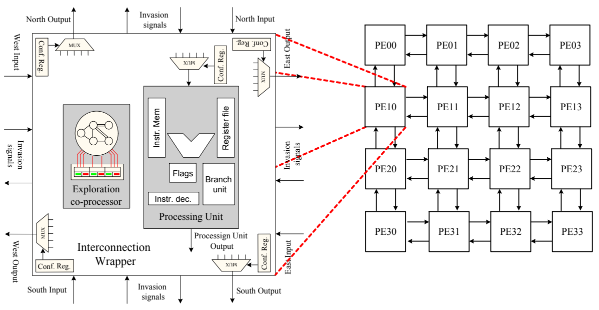

Fig. 1. Resource exploration controller for massively parallel processor architectures. One controller is integrated in each of the processing elements in a massively parallel processor array.

resources (*slave PEs*). Once the execution of all resources has finished, the number of captured resources can be altered by either issuing another exploration or retreat, respectively to increase or decrease the number of captured resources. In order to implement such functionality efficiently, an exploration controller may be integrated with each of processing elements as depicted in Fig. 1. The implementation of such a controller may vary according to the architectural requirements and constraints from a simple FSM-based implementation for area-limited architectures, as introduced in [6], to a programmable co-processor. Here, despite of the work in [6] where the controller was designed just for one-dimensional processor arrays, we have extended the controller to support exploration and reservation of a linear chain of connected PEs on general multi-dimensional architectures. The next section proposes different approaches for gathering resource explorations and for informing a configuration loader for infecting sets of captured PEs.

IV. Claim Collection

Once a resource exploration is performed, a configuration loader needs to be informed about the captured region specifications (number of PEs and their location). We call this information a "claim". In [6], a simple integer is incremented and rippled back to the master PE as the number of captured PEs after a successful exploration process. This mechanism could work perfectly for a simple linear array of processing elements, but for more complex architectures, like two dimensional coarse-grained reconfigurable arrays (CGRAs) [4], [7], such results should reflect not only the amount of captured PEs, but also their locations. In order to pass the geographical information about the captured PEs, two different types of approaches are proposed in the following: a centralized approach and three streaming-based (decentralized) approaches. **Centralized Approach:** In this approach, dedicated coordinate signals connect each PE to a resource manager which is responsible to detect the captured PEs and inform the configuration loader to configure the captured PEs with an appropriate program and interconnection topology. In order to achieve this vertical and horizontal signals might be shared among PEs of a same column/row to connect PEs in a same column/row to the resource manager respectively. The captured PEs use these vertical and horizontal signals to inform the resource manager about their locations. This approach can be implemented in two ways. In the first way, each PE enables its coordinate signals immediately after receiving an exploration request which enables to overlap the result collection time with the exploration time. Alternatively for the second approach, the exploration is completed first, then the configuration manager scans the array row-by-row and requests PEs to enable their vertical signals if they were captured. The first approach imposes less timing overhead to the system but the configuration manager should always snoop on the incoming signals. In the second approach, the configuration manager starts scanning just when an exploration has completed but at the expense of a slight timing overhead. In summary, the centralized approach imposes an insignificant timing overhead to the system but at the expense of hardware wiring cost, and the need for implementing a central resource manager.

Streaming-based (Decentralized) Approaches: In the following approaches, claims are gathered and communicated back to the master PE in a decentralized way. Subsequently, the master PE requests the configuration loader to reconfigure the captured PEs. In this case, a stream of information is built and sent to the master PE that includes information about the geographical location of the captured PEs. Here, we propose and compare three different streaming-based approaches.

Coordinate collection: In this solution, a claim consisting of information about the number of captured PEs during the exploration phase, followed by a list of coordinates of the captured PEs is sent from captured PEs to their master PE. Each PE appends its coordinate information at the end of a so-called claim stream and passes it to its capturer. This solution can even support simultaneous claim collection for different applications on a system. The disadvantage of this approach is its high data overhead, where for each PE, we need to include its coordinate information into the stream and depending on the array size, this data overhead may vary accordingly.

Directional collection: In this method, instead of appending a coordinates by the captured PEs, each PE adds just direction symbols to the stream indicating the direction of its own slaves (neighbors that the PE has captured), i.e., North, East, South, or West. In this way, less amount of data is gathered and transferred compared to the case of coordinate collection, but like the last method, this approach gathers claim information of a size proportional to the number of captured PEs.

Compressed directional collection: In this solution, a compression method is used to decrease the amount of transferred data. Here, instead of transmitting slave directions for each PE, the number of consecutive PEs captured in one direction is encoded and appended to the claim stream. Here, only if there is a change in the direction, a symbol will be placed in the stream showing the new direction of the captured PEs. In this way, each symbol includes two parts: the direction of the claimed PEs, i.e., North, East, South, or West, and the number of consecutive PEs in the specified direction. An example of this method is shown in Fig. **??**. In this example, the stream generation starts from the last PE in the captured domain, *PE03*, by sending $S_{clm} = 1N$, indicating that this PE is located at the north of its capturer and only one PE is captured in this direction. As there is a change of direction in *PE13*, a new symbol will be added at the end of the stream showing

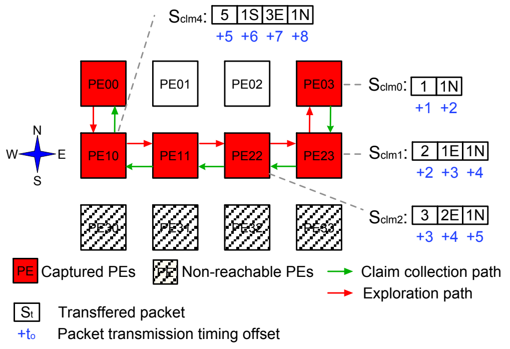

Fig. 2. Compressed directional result collection (streaming-based approach) for a simple captured domain where $PE00$ is the master PE. Note that the arrows just show the exploration steps. Each claim stream, transferred from a PE to its capturer, consists of a packet indicating the number of captured PEs, and comresssed directional symbols. Timing offsets annotated under each packet box, indicates the transmission timing offset that should be added to the claim collection start time ($T_{tr} = t_s + t_o$).

the geographical location of $PE13$ from $PE12$'s perspective, $S_{clm} = 1E1N$. As the direction remains unchanged for $PE12$, this PE increases the number of consecutive PEs in the last symbol, so the stream will be $S_{clm} = 2E1N$. This process continues until the main master PE, which receives the final stream $S_{clm} = 1S3E1N$. This approach is obviously superior for regular domain invasions with few directional changes. For more information about the proposed methods please see [8].

V. Cost Evaluation

The claim collection latency may vary depending on the architectural parameters and the claim collection mechanism that is implemented. In order to transfer exploration-related signals among PEs, a set of control signals is used which we call exploration data signals. The number and bit width of these signals can influence the timing cost of claim transmissions. Here, the proposed mechanisms are evaluated by considering system design parameters such as number of connected neighbors (N_{con}), number of rows/columns in a 2D processor array (N_{row}/N_{col}) and the bit width of control signals (D_{data}).

In the case of the proposed centralized approaches, the first solution does not impose any timing overhead. For the second one, we need at most N_{row} cycles to scan all rows, assuming each row can be scanned within one cycle. For the streaming-based approaches, the amount of stream data needs be calculated in order to estimate the timing cost of a claim collection phase. In the case of the coordinate collection method, where the PE coordinates are sent, the data size needed to encode each PE coordinate is $D_{coo} = \lceil \log_2(N_{row}) \rceil + \lceil \log_2(N_{col}) \rceil$ when encoding row and column coordinates separately. In our implementation a claim stream consists of a number indicating the total amount of captured PEs followed by each captured PE coordinate, we always reserve the first data item of the stream to transfer the captured PE count. It is assumed that this number can be presented by data bits ($\lceil \log_2(N_{row} \times N_{col}) \rceil \leq D_{data}$). Consequently, the total length of a claim stream (in bits) received by the master PE will be $D_{clm} = D_{data} + N_{cap} \times D_{coo}$ where N_{cap} is claim size (number of captured PEs). By calculating the maximum number of coordinates which can be placed in one packet, $N_{cpp} = \left\lfloor \frac{D_{data}}{D_{coo}} \right\rfloor$, the total number of coordinate packets received at the master PE can be calculated as $N_{coo} = 1 + \left\lceil \frac{N_{cap}}{N_{cpp}} \right\rceil$. By assuming that each PE starts informing its capturer PE immediately after receiving the first chunk of the result stream from its slaves, the total claim collection latency may be calculated as $T_{clm} = N_{coo} + N_{cap}$. In the directional collection approach, slave direction symbols are sent by the PEs besides the number of captured PEs. The size of direction symbols depends on the connectivity of the array architecture N_{con} (number of channels a PE is connected to) and is calculated by $D_{dir} = \lceil \log_2(N_{con}) \rceil$. The total size of a claim stream is then calculated as $D_{clm} = D_{data} + N_{cap} \times D_{dir}$. The maximum number of direction symbols per packet is $N_{dpp} = \left\lfloor \frac{D_{data}}{D_{dir}} \right\rfloor$. Consequently, by assuming the maximal number of stream packets received at the master PE, $N_{dir} = 1 + \left\lceil \frac{N_{cap}}{N_{dpp}} \right\rceil$, T_{clm} is calculated by $T_{clm} = N_{dir} + N_{cap}$.Each symbol of the compressed directional collection approach is constituted of two parts: a direction symbol and the number of consecutively captured PEs in the specified direction. The maximum number of consecutive PEs in one direction depends on the processor array size $N_{cons} = max[N_{row}, N_{col}]$, and the data size to send this number is $D_{cons} = \lceil \log_2(N_{cons}) \rceil$. The size of a direction change symbol, D_{dir}, is also calculated as explained for directional collection approach. Consequently, the total size of each symbol will be $D_{sym} = D_{dir} + D_{cons}$. The total size of the stream now depends on the number of symbols (N_{sym}) that are placed in a claim stream. As an example, assume a captured domain with the size of $N_{cap} = 10$, and different numbers of direction changes, and a set of direction symbols $C_{dir} = \{N, E, S, W\}$. Some examples of claim streams are: $S_{clm1} = 10E$ for a domain in a single direction, $S_{clm2} = 7E3S$ for a domain with one directional change, or $S_{clm3} = 1E1S1E1N5E1S$ with five directional changes. The longest stream occurs when the direction of the exploration is changed in every captured PE, meaning $N_{sym} = N_{cap}$, and the shortest stream happens when all of the captured PEs are in one straight line, meaning $N_{sym} = 1$. For a captured domain containing N_{sym} symbols, the total size of the stream is $D_s = N_{sym} \times D_{sym}$. With the maximum number of symbols per packet be given as $N_{spp} = \left\lfloor \frac{D_{data}}{D_{sym}} \right\rfloor$, the total number of compressed directional packets (received at the master PE) will be $N_{cdir} = 1 + \left\lceil \frac{N_{sym}}{N_{spp}} \right\rceil$, and the total claim collection latency may be approximated as $T_{clm} = N_{cdir} + N_{cap}$.

VI. Experimental Results

As a target platform, we consider a coarse-grained reconfigurable architecture consisting of an array of tightly-coupled lightweight reconfigurable processor elements [7] (see Fig. 1). These processor elements are single-threaded processing elements with very-long-instruction-word instruction set. Here, the exploration controller introduced in Section III is intergrated into a simulation model of this architecture to support resource explorations in four directions. It is worth to mention that the designed controller can be coupled to different kinds of MPSoC architectures such as RISC-based MPSoCs as well. In order to evaluate the functionality of the proposed streaming-based result collection mechanisms, the

	$N_{cap} = 4$	$N_{cap} = 8$	$N_{cap} = 12$
N_{coo}	2	3	4
N_{dir}	2	2	3
N_{cdir}	2	2.16	3

TABLE I
AVERAGE NUMBER OF PACKETS RECEIVED AT THE MASTER PE FOR A 4×4 ARRAY OF PES.

simulation platform is configured for different experimental scenarios. These scenarios can be categorized according to three different parameters such as array size, occupation rate ($O_{rate} = \frac{N_{occupied}}{N_{row} \times N_{col}}$) which results in randomized selection of some PEs (grouped into contiguous cascading blocks) as busy or non-available PEs, and N_{cap}, the number of PEs which are targeted to be captured in each experiment, where $N_{cap} = n \times N_{col}$ and $1 \leq n < N_{row}$. As a result of such resource explorations, e.g. a linear chain of PEs may be captured which can be used to execute signal processing and filtering applications, e.g. finite impulse response (FIR) filters, convolution calculations, or image filtering algorithms. Here our focus is on the performance of proposed decentralized methods, not on the application execution performance. The size of the control signal is 16 bits for all of the experiments ($D_{data} = 16$).

Tab. I shows the average number of the transferred claim packets on a 4×4 array of PEs for each of the proposed decentralized methods. The number of claimed PEs for the experiments varies from $N_{cap} = 4$ PEs to $N_{cap} = 12$ PEs and $10\% \leq O_{rate} \leq 90\%$.. As it can be seen, the number of transferred packets for coordinate collection and directional collection approaches grow linearly proportional to the number of captured PEs. But the compressed directional collection behaves differently when number of direction changes varies due to the occupation rate. In general for 4×4 arrays, the directional collection approach results in less packet transmission because due to the small array size, when the occupation rate is increased, the number of directional changes grows accordingly. Fig. 3 shows the average number of transferred packets on different array sizes, where in all cases $N_{cap} = \left\lfloor \frac{N_{row} \times N_{col}}{2} \right\rfloor$, and $10\% \leq O_{rate} \leq 50\%$. In order to have a fair comparison, the different approaches are compared according to the average amount of packets transmitted per captured PE. We call it *claim overhead rate*. As it can be seen, for the coordinate collection method, this rate is growing by increasing array size, meaning that we need to transfer more packets for bigger arrays, while capturing the same amount of PEs. In order to obtain the same performance we need to use wider control signals for bigger arrays. Conversely, the overhead rate for the directional approach remains unchanged which shows the its great scalability for different array sizes. The compressed directional collection shows its superiority over other approaches for the array sizes because there is more chance to gain consecutive PEs with less direction changes.

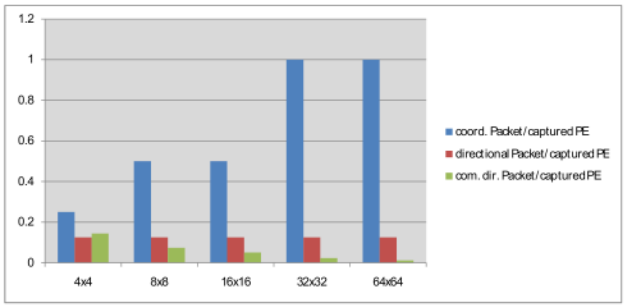

Fig. 3. Result overhead rate for different array sizes. The number of captured PEs is half of the array size and the result is given as an average over different utilization rates for each array size.

VII. CONCLUSION

This paper proposes two strategies for distributed resource exploration and reservation in massively parallel processor arrays. In the first category, a centralized approach was presented that was based on direct connections from processing elements in a massively parallel processor array to a central resource manager. In the second category, three decentralized streaming-based approaches were proposed. In order to include the geographical information of captured PEs, these approaches use coordinate information, directional information, or compressed directional information. A mathematical analysis has been presented to measure the timing and data overhead of each of the approaches. Our simulation results show that directional collection has better results for small arrays, but for big arrays the compressed directional method leads to more compact results and latencies.

VIII. ACKNOWLEDGEMENT

This work was partly supported by the German Research Foundation (DFG) as part of the Transregional Collaborative Research Centre "Invasive Computing" (SFB/TR 89).

REFERENCES

[1] K. Sankaralingam, R. Nagarajan, R. McDonald, R. Desikan, S. Drolia, M. Govindan, P. Gratz, D. Gulati, H. Hanson, C. Kim, H. Liu, N. Ranganathan, S. Sethumadhavan, S. Sharif, P. Shivakumar, S. Keckler, and D. Burger, "Distributed microarchitectural protocols in the trips prototype processor," in *Proceedings of the 39th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. Washington, DC, USA: IEEE Computer Society, Dec. 2006, pp. 480–491.

[2] P. Palatin, Y. Lhuillier, and O. Temam, "Capsule: Hardware-assisted parallel execution of component-based programs," in *Proceedings of the 39th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, Washington, DC, USA, 2006, pp. 247 –258.

[3] J. Resano, D. Mozos, and F. Catthoor, "A hybrid prefetch scheduling heuristic to minimize at run-time the reconfiguration overhead of dynamically reconfigurable hardware," in *Proceedings of Design, Automation and Test in Europe (DATE)*, vol. 1, Mar. 2005, pp. 106–111.

[4] V. Baumgarte, G. Ehlers, F. May, A. Nückel, M. Vorbach, and M. Weinhardt, "Pact xpp—a self-reconfigurable data processing architecture," *The Journal of Supercomputing*, vol. 26, pp. 167–184, 2003.

[5] J. Teich, J. Henkel, A. Herkersdorf, D. Schmitt-Landsiedel, W. Schröder-Preikschat, and G. Snelting, "Invasive Computing: An Overview," in *Multiprocessor System-on-Chip – Hardware Design and Tool Integration*, M. Hübner and J. Becker, Eds. Springer, Berlin, Heidelberg, 2011, pp. 241–268.

[6] F. Arifin, R. Membarth, A. Amouri, F. Hannig, and J. Teich, "FSM-Controlled Architectures for Linear Invasion," in *Proceedings of the IFIP/IEEE International Conference on Very Large Scale Integration (VLSI-SoC)*, Florianópolis, Brazil, Oct. 2009.

[7] D. Kissler, F. Hannig, A. Kupriyanov, and J. Teich, "A Highly Parameterizable Parallel Processor Array Architecture," in *Proceedings of the IEEE International Conference on Field Programmable Technology (FPT)*. Bangkok, Thailand: IEEE, Dec. 2006, pp. 105–112.

[8] V. Lari, F. Hannig, and J. Teich, "Distributed Resource Management in-Massively Parallel Processor Arrays," *Technical report at the department of computer science 12 (Hardware-Software-Co-Design)*, Feb. 2011.