International Journal of Networking and Computing – www.ijnc.org
ISSN 2185-2839 (print) ISSN 2185-2847 (online)
Volume 3, Number 1, pages 2–14, January 2013

Very Large-Scale Integrated Processor

Shigeyuki Takano
SANYO Semiconductor Co., Ltd.
Gunma, Japan

Received: June 4, 2012
Revised: October 3, 2012
Accepted: December 4, 2012
Communicated by Akihiro Fujiwara

Abstract

In the near future, improvements in semiconductor technology will allow thousands of resources to be implementable on chip. However, a limitation remains for both single large-scale processors and many-core processors. For single processors, this limitation arises from their design complexity, and regarding the many-core processors, an application is partitioned to several tasks and these partitioned tasks are mapped onto the cores. In this article, we propose a dynamic chip multiprocessor (CMP) model that consists of simple modules (realizing a low design complexity) and does not require the application partitioning since the scale of the processor is dynamically variable, looking like up or down scale on demand. This model is based on prior work on adaptive processors that can gather and release resources on chip to dynamically form a processor. The adaptive processor takes a linear topology that realizes a locality based placement and replacement using processing elements themselves through a stack shift of information on the linear topology of the processing element array. Therefore, for the scaling of the processor, a linear topology of the interconnection network has to support the stack shift before and after the up- or down-scaling. Therefore, we propose an interconnection network architecture called a dynamic channel segmentation distribution (dynamic CSD) network. In addition the linear topology must be folded on-chip into two-dimensional plane. We also propose a new conceptual topology and its cluster which is a unit of the new topology and is replicated on the chip. We analyzed the cost in terms of the available number of clusters (adaptive processors with a minimum scale) and delay in Manhattan-distance of the chip, as well as its peak Giga-Operations per Second (GOPS) across the process technology scaling.

Keywords: Fusion Core, Composable Processor, Adaptive Computing, Reconfigurable Computing.

1 Introduction

Thousands of compute and memory resources will be implementable on-chip in the near future through improvements in a semiconductor technology. However, we must face to the following design limitations in such the large-scale processor; Single large-scale processors have met with a design complexity issue in which a larger building block lengthens the critical path and requires an increase in power consumption [13]. The long critical path and greater power consumption of such processors create a difficulty in scaling the clock frequency and implementing high-level functionality to improve their performance. In previous decades, several back ground functional units have been integrated to improve instruction-level parallelism (ILP) by filling executable and issuable instructions into empty pipeline at the issue pipeline stage. Therefore, recent single processors have only several fraction of available area for integer and floating-point execution units. Rather than improving the ILP, a thread-level parallelism (TLP) was focused upon and improved through multi-threading in the 1990s [4]. These days, multi-core processors are a current trend. We now routinely use multi-core processors, and the number of cores will soon be "many".

Many-core processors are designed for improving the thread-level parallelism (TLP) across the cores, and for keeping the ILP in each core. However, each application has its own characteristic TLP and ILP. Therefore, a pre-fabricated chip multiprocessor (CMP) can not tolerate a wide range of applications. As is routinely conducted, the application must be optimized to the target computer system. Rather than optimizing the software, it is hot topic that aims to optimize the hardware through a dynamic reconfiguration of CMP to fit the processor to its applications [5] [2]. This approach is called a dynamic CMP. An alternative approach to fit the platform to its application involves the use of a reconfiguration technology such as field-programmable gate array (FPGA) devices [23] [22]. Recently, a conventional microprocessor has been merged with an FPGA into a single computing node [19] [20]. However, this processor type and its application designs are more complex. Such an application has to be partitioned to host program and many hardware tasks with hardware/software co-design tools. A higher application design workload makes it difficult for the reconfigurable computing to be used in mainstream technologies.

A heterogeneous massive parallel processor (MPP) has difficulty tolerating defects. As an optimization of the homogeneous MPP, a heterogeneous MPP technology will be applied after the verification of the major cores and a last phase in the market. Or rather than implementing a higher-level functionality, reconfiguring the datapath with middle grained functionality provides greater flexibility and more resource sharing for utilization. The issues concerning homogeneous and heterogeneous pre-fabricated MPPs will become end of a moot point as research begins on merging the CMP with a reconfigurable technology. A dynamic CMP has the potential to optimize the processor scale for running applications (tasks) dynamically.

For this research studied the dynamic CMP architecture to achieve efficient computing for thousands of on-chip resources. This architecture is called a very large-scale integrated (VLSI) processor. A VLSI processor is based on our prior work on, an adaptive processor (AP) which removes the necessity of partitioning, supports resource management and scheduling on chip, reduces the workload of the reconfiguration, and reduces the workload in designing a processor and its applications [14]. Because an AP does not require an instruction-set architecture in its basic model, we need to investigate how to interface between the VLSI processor and its application without an impact on the area and time overhead. In this paper, we discuss such an approach. Our approach to up- or down-scaling is simply to chain or unchain segmented interconnection networks using programming switches. The segmentation of the interconnection network is to prepare a set of minimum adaptive processor having sufficient resources, and to prepare to be possible to interconnect adjacent such APs. The AP can configure multiple application datapaths in a sequential configuration manner. The AP uses a linear topology. The linear array has to be folded efficiently into two-dimensional plane. To map an array to a two-dimensional array, we also propose a new topology to serve the scaling, which we call an S-topology. We show a simple and general model (concept model) of the S-topology and a model of the VLSI processor. Using the VLSI processor, we can obtain several benefits;

- **Processor Optimization.** An application has its own locality and dependency characteristics. We can obtain the optimal configuration of the processor that fits its particular characteristics through a reconfiguration. For example, a streaming application with a large (data) dependency will probably require more resources to configure its datapath. The application then requests the resources, and its datapath is configured to a large-scale AP. Application designers know the optimal amount of resources, and thus they should be able to control the reconfiguration through a certain methodology.

- **Balance between General-Purpose and Application-Specific.** Compared with application-specific processor (ASIP), the general-purpose processor does not achieve to performance of the ASIPs. We can coordinate a computational trade-off between an application and processor through a reconfiguration to optimal processor configuration. This balance depends on what point of the application the designer wants to optimize. It is probably coordination between clock cycle time and the number of resources that control the performance, throughput, area, and power consumption. At least the number of resources can be controlled by this technology.

- **Guard Data-Intensive Datapaths from Control-Intensive Datapaths.** Occasionally, a control-flow flashes the processor pipeline and decreases the performance and throughput. Regarding the AP, the control-flow breaks a regularly reconfiguring datapath, and creates unpredictable swap-in and -out of the object. The basic blocks can perform with a small amount of overhead and without interfering with the execution of other blocks, if the basic blocks, which are partitioned by the control-flow, are mapped to the VLSI processor. The isolated basic block processors can communicate through an inter-processor communication that activates following basic block(s).
- **Defect Tolerance.** Scaling to hundreds or thousands of processor elements and memory blocks on chip will increase the number of defects. Through the VLSI processor architecture, the failing AP can be removed from the system. For example, when four APs are used on chip and they can be fused into one large-scale processor, two medium-scale processors, and four small-scale processors. When a second AP fail, the first processor can become a small-scale processor, the third and fourth processors can be fused into the a medium-scale processor or split into two small-scale processors.

The next section shows our prior work on an adaptive processor. Section 3 explains and discusses the basic S-topology. Section 4 shows the costs in terms of area and delay, and assesses the peak performances derived from the delay. Other work related to this topic is then discussed in section 5, and some concluding remarks are given in the final section.

2 Adaptive Processor

As we move toward thousands of processing and memory elements on a single chip, one of the most important topics, essential for achieving a peak performance, is resource management and scheduling. The CMP does not support resource management and scheduling on chip. The larger scale of a many-core processor will easily result in a larger gap between the peak and effective performances, probably causing a delay of many cycles for the managing and scheduling of resources. The alternative is that greater optimization effort will be required. We know that the most frequently used operations should be on chip; therefore, we implemented such functionality in this manner. This section explains such a processor called an adaptive processor, which was designed to reduce the workload of the processor and application designs, resource management and scheduling, and reconfiguration.

2.1 Object and 2-Level Configuration

A processing element called a physical object performs its operation as defined by the configuration data. Such configuration data is called local configuration data. The pair of initial data and local configuration data is called a logical object, and logical object binded on the physical object is called an object. The logical object can move on array of the physical object explained in later. To configure an application datapath, chaining between operators is defined through the global configuration data which consists of a sink object ID and source IDs. Therefore, in a global configuration data stream, the dependency is represented by the ID.

2.2 Processor Pipeline

The adaptive processor has the following pipeline stages:

1. **Pointer Update** Pipeline stage used to update a pointer addressing an element of the global configuration data stream. This stage is independent from the following pipeline stages.

2. **Request Fetch** Pipeline stage used to fetch an element of the global configuration data. This stage is similar to the instruction fetch pipeline stage in conventional pipelined processors.
3. **Request Evaluation** Pipeline stage is used to evaluate a request with the fetched element. An evaluation of a memory access request is carried out at this stage.
4. **Request** Pipeline stage is used to request resources (necessary objects). Global configuration data stream for object cache-miss is inserted at this stage.
5. **Acquirement** Pipeline stage is used to acquire resources for the request. Routing is performed during this pipeline stage using an acquirement signal from special registers called a working-set register file (WSRF) for maintain the acquired elements.

After the acquirement, the objects are free from control. An object is released by receiving and firing release token(s) from the preceding object(s).

2.3 Configuring an Application

Figure 1 shows the procedure used to configure a part of the application datapath. The datapath consists of a set of objects. The figure shows the requested objects, the procedure used for chaining between objects, and the termination of their operation. At the request pipeline stage, necessary resources (logical objects) are searched in the processor. The "hit" object acknowledges the hit and activates the execution fabric in the physical object. In the next cycle, the acquirement pipeline stage, the object will receive an acquirement signal from a WSRF. The acquirement signal indicates which communication port to use for the chaining between objects. When it is an object cache-miss, its logical object(s) is loaded from the library in the memory blocks to a configuration buffer object(s). After loading all requested but cache-miss logical object(s), the processor forces a stack shift from the top of the stack to the bottom of the stack to enter the loaded logical object(s) into the object space. After logical objects have been entered, the objects are requested again and will be chained (acquired).

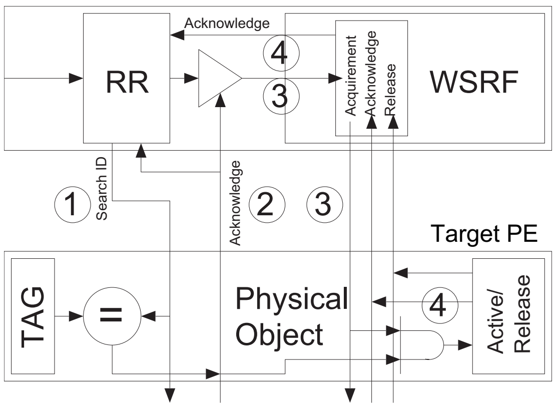

Figure 1: Configuration Procedure on Pipeline [14]

2.4 Stack Shift and Dependency

An array of physical objects composes a stack structure. The stack structure creates a deterministic and locality based placement; this placement is always on the top of the stack. Because a stack shift sorts the objects in the array, a replacement, based on an LRU algorithm, is easily implemented, and objects close to the bottom of the stack are candidates for the replacement.

A study on stack algorithms showed a relationship between the stack distance and cache hit rate [11]. The stack distance is the distance from the top of the stack to the cache hit location (physical object). To make a hit always occur, the stack distance $\triangle$ has to be less than or equal to C, where C is the capacity of the cache, namely the array size for the adaptive processor.

An element of the global configuration data stream requests resources. After acquirement of the request, the resources are on chip and chained among them. The element simply expresses an object ID. Therefore, the stream shows the dependency used for the chaining. The stack distance is equivalent to the dependency distance in the CACHE model [14]. The dependency distance can be observed by an object code showing the object IDs.

2.5 Virtual Hardware

An unused object should be swapped out to a memory block to make room for a newly requested object(s). This replacement is equivalent to the write-back policy of conventional cache memory. When it is an object cache-miss, cache missed object(s) is loaded, and replaceable object(s) is stored if necessary. The replacement is scheduled using a special interconnection network composing a scheduling table [14]. The virtual hardware is supported when the processor works on completely scalar operations. When an operation involves streaming, the reconfigured datapath has to be smaller than the capacity C, since the streaming does not allow swapping out part of the datapath.

2.6 Channel Segmentation Distribution Model

The basic AP uses a global interconnection network for the chaining. This network was not considered in previous works, demonstrating the CACHE model. The global interconnection network is suitable only for a small number of physical objects (a small area). We therefore need to resolve this limitation. This section proposes a global interconnection network architecture, and evaluates the ability using our developed functional simulator.

2.6.1 Chaining Processors for Scaling

In general the number of channels used for global interconnection network chaining between a sink and source objects is linearly increased by the number of physical objects. A channel segmentation distribution (CSD) of the AP has the potential to introduce a constant number of channels. The scaling of the AP simply chains the segmented global interconnection networks, used for finding LRU object(s), the stack shift, and so on. Cache hit detection can be centrally processed on the WSRF instead of searching in the array; however, sending a request to the object space is still necessary for detecting and waking up the cache hit objects to prepare for establishing a communication between the sink and source objects. Searching in WSRFs can be performed in parallel.

2.6.2 Extended Model of CSD Network

An approach to allocate a channel to the chaining is suitable only for static configuration, which we have examined. This subsection discusses and proposes a method to extend the basic CSD network. For a CSD network, the channel has to be selected dynamically. Our approach is to make a dynamic CSD network with chaining or unchaining in which each channel is completely segmented with a single hop. Segments are chained at the initial state, and unchained through a routing procedure. Figure 2 shows the very simplified logic circuit used for this approach.

The source object broadcasts a request signal to every channel. The signal passes through an interconnection network that is also segmented with one hop, where the default state is "chained". The sink object has a priority encoder that decides which channel is used for the request, several requests can come through surviving such as already used for other communication (chaining) on each channel. A grant signal from the encoder is checked by the sink object which can know the routing request. The grant signal is stored in a memory cell that controls the unchaining of the request network and to gate data from the channel to the sink object. The grant signal is sent back to the source object as an acknowledgement. This approach is capable of stack-shifting from the top to the bottom of the stack. Therefore, the decision to select the channel, send the channel number, and the acquire a signal are unnecessary for this sequence.

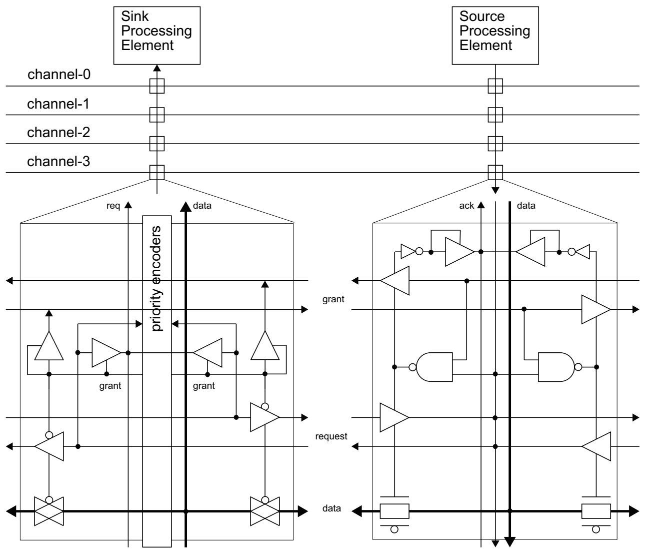

Figure 2: Dynamic CSD Network

An object including a memory unit is treated as out of the stack, and therefore the interconnection network has to be reachable to these objects. Thus, when trying to load from or store to the memory object, it takes long delay. We must take this as the worst case delay. One approach for implementing the stack structure with this delay, is to take a library using small number of metal layers which generates relatively longer delay. This approach aims at making the delay equal to the worst case delay. The physical object can have a smaller area, and thus a higher density, and therefore greater number of physical objects can be in the same area.

We developed a functional CSD simulator for the evaluation. Figure 3 shows the evaluation results of a one-source model (not a two-source model), and how many channels are used in a random datapath configuration. N_{object} is the number of physical objects. The figure shows the locality versus the number of channels used. A random request of a sink object and a locality based request of a source object were used. Regarding the source object ID, the preceding sink object ID and an offset are used, and therefore by controlling the offset we can generate a random configuration with the locality, where a higher locality takes a very small number or is equal to zero. Thus, the shape of each curve comes from the locality calculation (ID generation) using a random number generation. The left most plots have a higher locality that uses a smaller number of channels in general. The figure shows that N_{object} channels were not used, and $N_{object}/2$ channels are sufficient for the random datapath. Although the necessity of a fan-out (broadcast) requires more channels, i.e., up to N_{object} channels, we can allocate the remaining channels to the fan-out.

This approach must consider how much of an area reduction is acceptable to provide sufficient routability.

2.7 Summary

The processor pipeline serves the resource management and includes the placement and routing in the pipeline stage. The placement is always on the top of the stack, and the stack structure serves the LRU replacement and object caching. The dependency configures the application datapath, and therefore, the global configuration data simply represents the dependencies. The dependency distance is a key for efficient processing. We need to take care that the distance be no larger than the capacity to avoid making an object cache miss. The dynamic CSD approach was proposed to reduce the area required. The number of channels required for a dynamic CSD network is determined by the spatial locality, for deciding the dependency distance, the temporal locality indicating how frequently communicated, and the communication orders to consume the channels that decides the

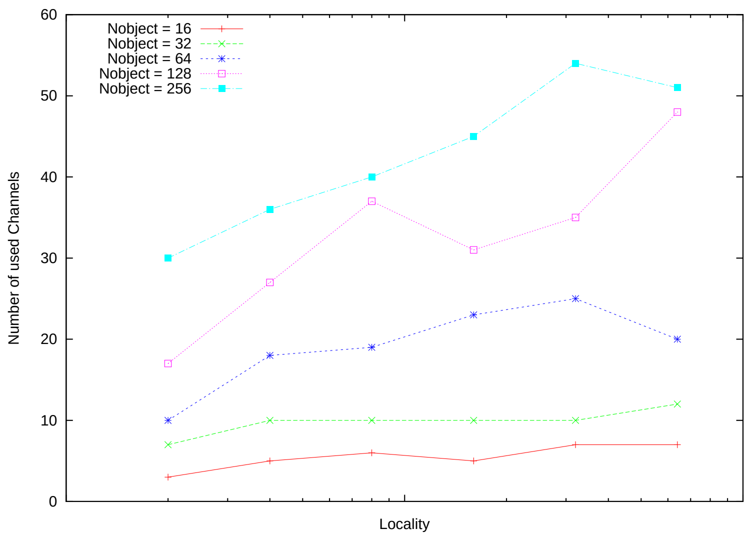

Figure 3: Locality versus Number of Used Channels

communication path allocation on channels of the dynamic CSD network. The routability is a trade off for the area requirement.

3 2D Arrangement for Linear Array

The AP uses a linear array topology to compose a stack structure. To map the array to a two-dimensional array, a new scalable topology, which we call an S-topology is also explained in this section. In addition, how to configure or up/down scaling the AP is discussed and studied and discussed in this section.

3.1 S-Topology

A topology has to have the following properties:

1. The topology has to be hierarchical or fractal. For fair placement on any processing element, a hierarchical or fractal topology is necessary. In addition it helps making a simple structure for scaling.

2. The array has to have a minimum number of patterns for the layout. A minimum clustering of processing (and memory) elements allows greater flexibility and reconfigurability.

3. The chain/unchain switch point has to have a regular pattern. Such a pattern provides predictability for controlling and routing. This supports the placement and routing on a very large-scale array.

Figure 4 (a) shows the S-topology. The cluster shown in Figure 4 (b) is simply replicated. Although the topology in the figure has a swinging path, this is only a conceptual image, and the actual path is straight, as shown in Figure 4 (c). Each AP has a stack structure and thus a linear topology. The linear network is folded into a 2D arrangement as shown in Figure 4 (c). Although the figure shows a unidirectional, it is a stack shift direction, and a bidirectional path is possible using the proposed dynamic CSD architecture. By applying the dynamic CSD and a modular structure that does not require a large number of metal wire layers, the folded linear network can be placed on the higher metal layers in a fashion similar to that of a recent many-core processor [8]. The

S-topology network supports the ability to unchain (split) the array into any arbitrary shape that may be formed by connecting the clusters as shown in Figure 5. The shape can form a ring topology in a 2D array.

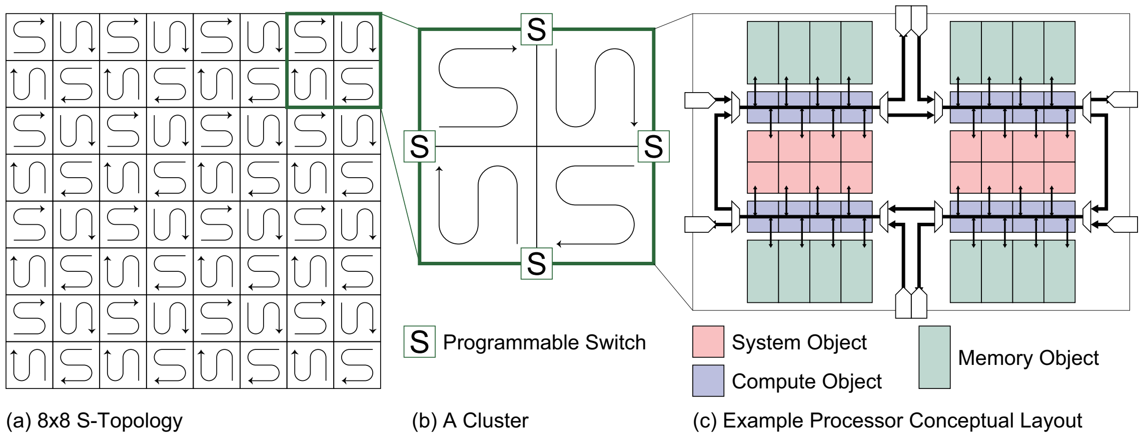

Figure 4: S-Topology

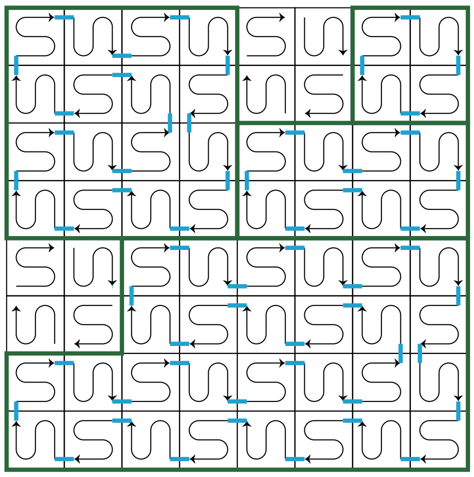

Figure 5: Rings on S-Topology

3.2 Programmable Switch

Any arbitrarily shaped region can be configured in the array using the programmable switches. Figures 6 (b) and (c) show the basic architecture. The box shown in the figure is a programming register. Figures 6 (b) and (c) show a programmable switch for a unidirectional path of the stack shift interconnection network and a programmable switch for a bidirectional path of the chain interconnection network. The default status of programmable switches is a "unchained". We can implement the VLSI processor using a die-stacking (chip-on-chip) by connecting the bottom and top side dies as shown in Figure 6 (d).

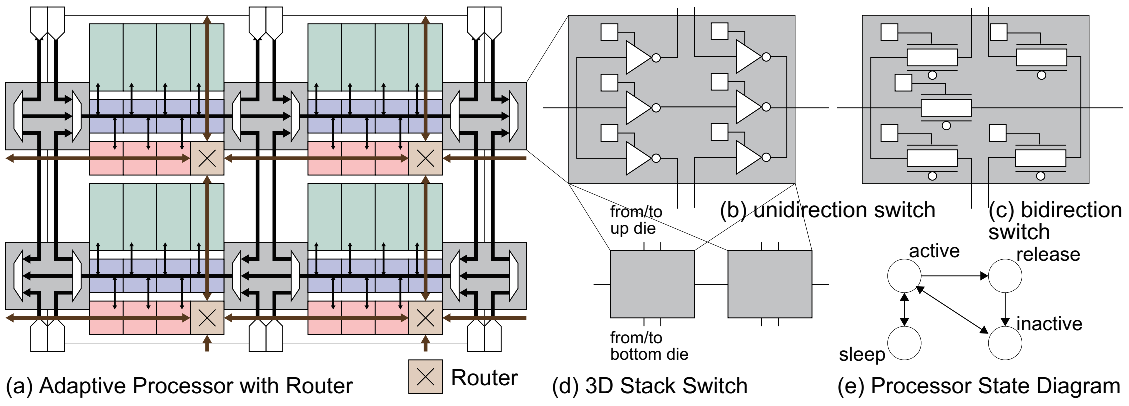

Figure 6: Router and Programmable Switch

3.3 Scaling Operations

To configure an AP with the necessary scale, we should first configure the processor at an executable scale (a minimum requirement for an application task) by gathering the clusters (resources). Up scaling can then be performed. The scaling is done by programming the switches through wormhole routing using on-chip routers, and thus we can reconfigure the processor by storing the appropriate configuration data to appropriate switch.

Figure 6 (e) shows a basic state diagram consisting of release, sleep, active, and inactive states. First the processor starts from and ends with the release state that is not used and allocated. After programming the switches in a minimum AP, the processor turns into an inactive state that is ready to execute but not read and write protected from others. Either a timer, or read and write protections in the scaled region are set, and te region is invoked as the scaled active AP. The active processor can be in an inactive state by clearing the read and/or write protection. In an inactive state, others can access its memory blocks. Thus storing a global configuration data, storing objects into libraries, spilling and filling of data in the memory block are done in this state. Fetching the global configuration data depends on the application. The sleep state is ready to execute and is read- and write-protected from others. In addition, the global configuration data is not fetched in this state. The active scaled AP can sleep and wait for an event by setting the timer, or wait for an event from inside. Therefore, the sleep state can be used for processor-level synchronization.

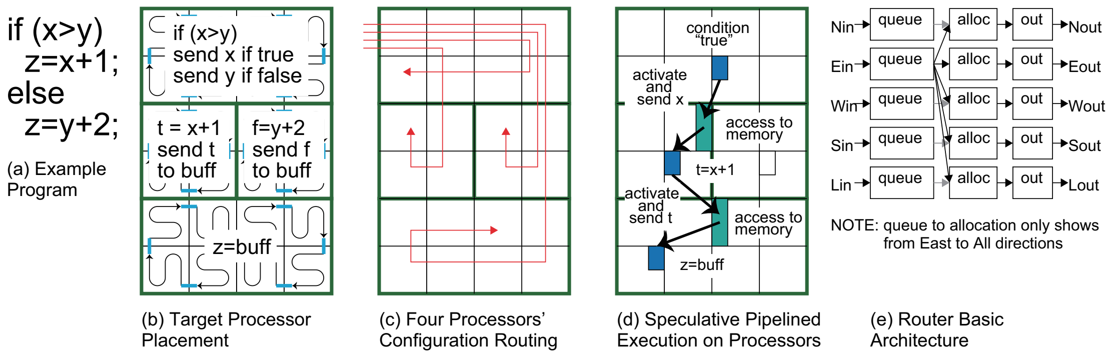

Figure 7: Example Processor Configuration, Its Routing, and Execution

Figure 7 shows a very simple example to show the scaling procedure and execution. The application can be partitioned into four atomic blocks as shown in Figure 7 (b). The atomic blocks can be a scaled AP. Another processor, which may be a preceding atomic block or supervisor processor configures the four processors. An in-order configuration may perform a spatially local placement as shown in Figure 7 (b). The configuration is based on wormhole routing as shown in Figure 7 (c). Wormhole routing is used to store a reservation flag at each programmable switch to avoid a resource (cluster) allocation conflict among the scaling configurations. After the configuration, each processor is initially in an inactive state. The preceding processor accesses and writes data to the memory block of following processor (shown in Figure 7 (d)). The first processor sends data to either the second or third processor depending on the condition. The second or third processor is activated and sends the result to the fourth processor. The fourth processor receives the appropriate data during its inactive state. This can be a pipelined execution through multiple processors. In general, a conditional execution can break the regular partial reconfiguration of the scaled AP and can have a negative impact. By isolating the application to basic blocks that are independent of each other regarding their control flow, this example does not have the negative impact. By isolating the basic blocks, this example does not have such an impact.

3.4 Inter-Processor Communication

An on-chip router used for inter-processor communication can be applied to the scaling. Figure 7 (c) shows the proposed reconfiguration methodology for the up-scaling. We use a wormhole routing for the up and down scalings. The execution uses an inactive state, whereas the preceding processor makes the processor active. Before activation, the processor stores sending data to memory block as shown in Figure 7 (d). Figure 7 (e) shows the current router architecture under development. The down-scale made is possible with wormhole routing along with the unidirectional routing by clearing active state, turns to be a release from the active state.

4 Cost Assessments

4.1 VLSI Processor

Table 1 lists the area requirements for a physical object, obtained from [12]. A general-purpose compute fabric includes 64bit floating-point and ALU modules. Because the reference for these area requirements does not include dividers, we used the weight values estimated from [17] to calculate the raw area of the dividers. Table 2 lists the area requirements for the memory block. ALU-II is used for the vector length, hardware-loop, and so on. An instruction register is used for a sequencer object. We used the configuration of 64KB SRAM, trading off for an area. The total memory block takes approximately twice the area of the physical object. Table 3 lists an area requirement for the control objects, which area is assessed only for the registers.

Modules	Process [um]	Area [λ^2]
64b fMul, fAdd	0.25	1.35×10^8
64b fDiv	0.25	0.21×10^8
64b iMul + iALU/Shift	0.25	2.90×10^8
64b iDiv	0.25	0.81×10^8
64b Register x6	0.25	5.36×10^6
Total		5.32×10^8

Table 1: Physical Object Area Requirement

Table 4 shows how many the APs having 16 physical objects and 16 memory objects are available. The silicon die area is held constant at 1 cm^2 which is ordinary chip area. The λ^2 area is calculated using rc-delay which is referenced from [21]. A global wire delay is calculated as the square root of λ^2 (the total area of the physical object, and is shown in Table 4). We obtained the peak GOPS (Giga Operations per Second) values excluding the load and store streams, as shown in Table 4, which are assessed from the global wire delays as a critical delay used for chaining between the

Modules	Process [um]	Area [λ^2]
32b ALU-I	0.25	0.86×10^8
16b ALU-II x4	0.21	1.72×10^8
Instruction Reg.	0.25	1.79×10^6
64b Register x2	0.25	1.79×10^6
64KB SRAM	0.35	7.13×10^8
Total		9.75×10^8

Table 2: Memory Block Area Requirement

Modules	Process [um]	Area [λ^2]
64b x40 Reg. in WSRF	0.25	35.7×10^6
64b x6 Reg. in CMH	0.25	5.36×10^6
64b x8 Reg. x2 in RR	0.25	14.3×10^6
64b Reg. in IRR x16	0.25	14.3×10^6
64b x2 Reg. in CFB x3	0.25	5.36×10^6
Total		75.2×10^6

Table 3: Control Objects Area Requirement

memory block and the physical object since the memory block can not be relocated, therefore a global network is still required. The area ratio of physical to memory objects is 1 : 2, and thus less than a 33% chip area is allocated to the FPUs. We can coordinate the number of FPUs and memories, and more GOPS is available if we optimize for more FPUs and less memory blocks. The VLSI processor is competitive with traditional GPUs, which takes at least three-times the area. We obtained three-times number of FPUs and memory blocks on this area size, although a delay negates the clock cycle time improvement.

Year	Process [nm]	Available # of APs	Wire-Delay [nsec]	Peak GOPS
2010	45	12	1.08	178
2011	40	16	1.21	211
2012	36	21	1.21	276
2013	32	24	1.43	269
2014	28	34	1.58	345
2015	25	41	1.56	432

Table 4: Number of APs, Wire Delay, and Peak GOPS

5 Related Work

A tile processor maps an application to a tile array, that the tile includes a processor and a router [4] [1] [3] [6]. The project in [4] is the first tile processor where each processor has bypass paths connecting it to its neighbors [7]. This type of architecture requires a partitioning application for working-sets (tasks) consisting of a program and data since the tiles are not scaled. The unchained approach also requires synchronization between tasks, and thus an inter-processor communication methodology is also required. Moreover, the placement (configuration) is probably static. As our approach, is scaling based, it eliminates the need to compile a group of dedicated working-sets. Objects form the application datapath. An application compiler needs to simply take care of the linear array size to fit the application datapath to the fused region, enabling streaming on the datapath.

Core fusion [5] and composable processors [2] have recently been proposed as scalable methods. Core fusion has an advantage in that the processor does not require a compiler to schedule the fusing and splitting instructions; however, the scalability is limited to a fusion no higher than eight-issue processor. Splitting and fusion instructions are in its instruction set architecture. A composable lightweight processor has the advantage of a reconfiguration to a large scale core: however, it requires a special compiler for the scheduling. In addition, the data-flow processing of a conventional processor philosophy results in an inefficient data-flow: the commitment of the instructions is delayed until the completion of the data flow on the critical path. A large-scale array may potentially have a long critical path. The VLSI processor uses unchaining and chaining; there are no specific instructions. In addition, there are no specific procedures used for chaining and unchaining, and it simply requires routing and storing the data set used in an ordinary communication. The scalability is limited only by the format of the configuration data and wire delay. A resource is released by firing the release tokens. This technique reduces the idling time as rapidly as possible.

A ring topology has been recently used for multi-core processors [15] [8]. The topology supports relocatability for flexible placement. Its latency is increased by the number of cores. This technique is scalable for a small number of cores. The combination of a modular structure and this topology allows for a flexible configuration, and thus the number of cores may be changed at the design time without a significant impact on the layout. However, a mesh topology has recently become a popular alternative [1] [3] [6]. This topology is very simple and completely scalable and relocatable. It also has an abundant bisection bandwidth. Though it has the freedom of placement, a host system has to manage the placement, routing, replacement, and defragmentation. As previously shown, the ring topology can be implemented on the S-topology. The VLSI processor is manageable.

6 Conclusion

This paper proposed a scalable processor, called a very large-scale integrated (VLSI) processor, that can up- and down-scale the datapath of the adaptive processor to dynamically configure and reconfigure the processor for the appropriate compute and memory resources. Up- or down-scaling is simply to chain or unchain between the segmented interconnection networks. The scaling does not require a dedicated instruction, and is to simply store the appropriate configuration data to the appropriate programmable switch with a wormhole reconfiguration (routing). There is no specific logic circuit required for the scaling. Therefore, the area cost is very low. The adaptive processor uses a linear topology to form a stack structure. To map the linear array to a two-dimensional array, we also proposed the S-topology. The dynamic CSD network was proposed and applied to the VLSI processor. The dynamic CSD network can reduce the area requirement. A reduction in the number of channels must be carefully performed by processor architects because the number of channels determines the routability. The costs in terms of area requirements and delays, and peak performances were assessed in this study. The performance of a pure 64bit 276 GOPS can be achieved in a typical 1 cm^2 area without both of SIMD features and fused operations, on current process technology.

References

[1] A.T. Tran et al., "A GALS many-core heterogeneous DSP platform with source-synchronous on-chip interconnection network", 3rd ACM/IEEE International Symposium on Networks-on-Chip, pp.214-223, 2009

[2] Changkyu Kim et al., "Composable Lightweight Processors," Proceedings of the 40th Annual IEEE/ACM International Symposium on Microarchitecture, MICRO 2007, pp.381-394, 2007

[3] David Wentzlaff et al., "On-Chip Interconnection Architecture of the Tile Processor", IEEE Micro, vol.27, no.5, pp.15-31, 2007

[4] Elliot Waingold et al., "Baring it all to Software: Raw Machines", IEEE Computer, pp.86-93, 1997

[5] Engin İpek et al., "Core Fusion: Accommodating Software Diversity in Chip Multiprocessors", Proceedings of the International Symposium on Computer Architecture, ISCA 2007, 2007

[6] J. Howard et al., "A 48-Core IA-32 Processor in 45 nm CMOS Using On-Die Message-Passing and DVFS for Performance and Power Scaling" IEEE Journal of Solid-State Circuits, vol.46, no.1, pp.173-183, 2011

[7] Michael Bedford Taylor et al., "Scalar Operand Networks: On-chip Interconnect for ILP in Partitioned Architectures", Proceedings of the International Symposium on High Performance Computer Architecture, February 2003

[8] M. Yuffe et al., "A fully integrated multi-CPU, GPU and memory controller 32nm processor", Proceedings of the 2011 IEEE International Solid-State Circuits Conference Digest of Technical Papers (ISSCC) 2011

[9] Peter J. Denning, "The Working Set Model for Program Behavior", Communications of the ACM, pp.323-333, vol.11, no.5, 1968

[10] Richard C. Holt, "Some Deadlock Properties of Computer Systems", ACM Computing Surveys, pp.179-196, vol.4, no.3, 1972

[11] R. L. Mattson et al., "Evaluation Techniques for Storage Hierarchies", IBM Systems Journal, pp.78-117, vol.9, no.2, 1970

[12] S. Gupta, et al., "Technology Independent Area and Delay Estimations for Microprocessor Building Blocks", The University of Texas at Austin, Department of Computer Sciences, Technical Report TR-00-05, February, 2001

[13] Subbarao Palacharla, "Complexity-Effective Superscalar Processors", Ph.D. Dissertation Thesis, University of Wisconsin, Madison, 1998

[14] Shigeyuki Takano, "Design and Analysis of Adaptive Processor", ACM Transactions on Reconfigurable Technology and Systems, vol.5, no.1, 2012

[15] Thomas William Ainsworth et al., "Characterizing the Cell EIB On-Chip Network", IEEE Micro, vol.27, no.5, pp.6-14, 2007

[16] Vaughn Betz et al., "Architecture and CAD for Deep-Submicron FPGAs", Kluwer Academic Publishers, 1999

[17] Venkatraman Govindaraju et al., "Dynamically Specialized Datapaths for Energy Efficient Computing", In Proceedings of 17th International Conference on High Performance Computer Architecture, 2011

[18] W.J. Dally, "Virtual-channel flow control," IEEE Transactions on Parallel and Distributed Systems, vol.3, no.2, pp.194-205, Mar 1992

[19] Zynq-7000 EPP Product Brief, Xilinx, Inc., 2011

[20] User-Customizable ARM-Based SoC FPGAs for Next-Generation Embedded Systems, Altera corp., 2011

[21] International Technology for Roadmap Semiconductors (ITRS), http://www.itrs.net/Links/2007ITRS/Home2007.htm

[22] IEEE International Symposium on Field-Programmable Custom Computing Machines, http://www.fccm.org

[23] International Conference on Field Programmable Logic and Applications, http://www.fpl.org