A design scheme for a reconfigurable accelerator implemented by single-flux quantum circuits

Farhad Mehdipour, a,b Hiroaki Honda b, Koji Inoue a, Hiroshi Kataoka a, Kazuaki Murakami a,b

a School of Information Science and Electrical Engineering, Department of Informatics, Kyushu University, Fukuoka, Japan
b Institute of Systems, Information Technologies and Nanotechnologies, Fukuoka, Japan

ARTICLE INFO

Article history:
Received 1 April 2010
Received in revised form 16 July 2010
Available online 11 August 2010

Keywords:
Reconfigurable accelerator
Single-flux quantum
Data flow graph
Placement and routing

ABSTRACT

A large-scale reconfigurable data-path processor (LSRDP) implemented by single-flux quantum (SFQ) circuits is introduced which is integrated to a general purpose processor to accelerate data flow graphs (DFGs) extracted from scientific applications. A number of applications are discovered and analyzed throughout the LSRDP design procedure. Various design steps and particularly the DFG mapping process are discussed and our techniques for optimizing the area of accelerator will be presented as well. Different design alternatives are examined through exploring the LSRDP design space and an appropriate architecture is determined for the accelerator. Primary experiments demonstrate capability of the designed architecture to achieve performance values up to 210 Gflops for attempted applications.

© 2010 Elsevier B.V. All rights reserved.

1. Introduction

Complicated computations and simulations necessitate providing high computational power for individual researchers. Although, recent advances on chip design and fabrication provide the possibility for producing high-performance computers, there is still a high demand to meet the required performance for specific applications. In a hybrid reconfigurable processor architecture, reconfigurable hardware as an accelerator can significantly relieve the burden of the main processor in computation-intensive part of applications.

Undoubtedly, there are some serious barriers in realizing powerful computing systems using recent finer CMOS technologies. As an alternative to CMOS circuits, single-flux quantum circuits seem suitable due to featuring high-speed transmission, high-speed signal transmission, low-power consumption as well as low-heat radiation. The basic component of SFQ digital circuits is a superconducting loop with Josephson junctions. A single-flux quantum appears as a 1 mV extremely low-width (in the range of Pico-seconds) and carries information in the very high-speed (up to the light speed) through the circuit, while in CMOS circuits the speed is limited due to the time required for charging/discharging capacitors [8]. Further, SFQ circuits operate based on pulse logic and each gate is a clocked gate and has a function of a latch. Hence, latches are implemented without additional costs and this makes SFQ circuits suitable for pipeline processing on streaming data. On the contrary, SFQ circuits are not suitable for processing of feedback loops and conditional branches. One issue in implementing very large-scale high-performance circuits by means of semiconductor circuits is the heat radiation and difficulty in high-density packaging. Unlike CMOS circuits, a SFQ circuit includes extremely lower power consumption and also a very smaller area compared to its CMOS counterpart.

Nowadays, memory wall problem is a well-known problem which comes from the gap between operating speed of a processor and that of memory. In other word, the memory bandwidth required for data communication between general purpose processor (GPP) and the memory is not enough to fill in this gap. In SFQ-circuits this problem might be more crucial, because implementing large-scale superconducting random-access memory is difficult while SFQ processing units can operate so fast. Most of accesses to memory are due to reading/writing the intermediate data which associates with spill code. An appropriate architecture capable of reducing the impact of spill code can alleviate the impact of memory wall problem as well.

To overcome abovementioned issues in implementing very large-scale accelerators, a new architecture has been introduced which consists of a CMOS general purpose processor, a memory and a single-flux quantum (SFQ)-based reconfigurable large-scale data-path processor (SFQ-LSRDP) as an accelerator [18]. Hundreds of processing elements (PEs) are arranged over a two-dimensional
array so that the output of each PE can be forwarded to the inputs of one or more PEs through the flexible operand routing networks (ORNs). The proposed architecture is expected to be a high-performance desk-side computer with low-power consumption and is suitable for execution of scientific applications demanding massive computations.

Since the LSRDP as a main component of the target architecture is implemented by SFQ circuits; obviously, it can mostly address the abovementioned issues originated from CMOS technology. Further, due to pipelined architecture of the LSRDP, the cascaded PEs can generate the final result without temporally memorizing intermediate data, therefore the number of load/store operations will be reduced and the memory-bandwidth pressure can be relieved. In addition, through providing the LSRDP with dynamic reconfiguration facilities it would be possible to program processing elements and routing resources at run-time. Hence, the LSRDP is an adaptable architecture as well.

The basic design procedure and preliminary results have been introduced in our previous report [10]. In this paper, more details will be given about the proposed ideas. Besides, the final outcome of the design procedure targeting the overall accelerator’s area, optimization achieved through a design space exploration will be presented in this version. Effect of design parameters on the area of operational and routing components of the accelerator will be analytically discussed. This analysis help us to explore the design space and obtain an optimized design in terms of the resource usage. A preliminary performance evaluation result will be reported as well.

In the rest of this paper, Section 2 explains related work from the hardware and software points of view. The basic specifications of the LSRDP architecture are highlighted in Section 3. Section 4 discusses about the design procedure and the relating tool chain. A benchmark set of computational-intensive scientific applications are introduced which are attempted for designing the target hardware. Within the LSRDP design procedure, placing DFG nodes including input, operations and output nodes on appropriate locations are discussed in this section as well. Further, an algorithm based on proximity factor of input node pairs as well as a connection length measurement technique are presented in Section 4. In Section 5, the design space exploration for examining variety of architectures for the LSRDP are discussed. The experimental results will be shown in Section 6 and finally, Section 7 concludes the paper.

2. Related work

Today’s technology allows increasingly more devices on a chip, so coarse-grained reconfigurable architectures (CGRAs) emerged as programmable platform in complex single-on-chips (SoCs) [7]. Typically, a CGRA has identical processing elements (PES), with variety of functionalities and interconnection topologies for connecting PEs. Register files (RFs) are distributed throughout the CGRA to hold temporary values and are shared by a subset of FUs. Morphosys [9], ADRES [11] and Silicon Hive [16] are examples of CGRAs.

A number of special-purpose computing systems have been proposed for efficient execution of scientific applications such as molecular orbital calculations. EHPC/ERIC is a parallel computer which has been developed particularly for this calculation using TSMC 0.13 μm CMOS technology [12]. It has a large number of special-purpose computing units to achieve a high level of parallelism. The achievable performance in this machine is strongly influenced by the memory wall problem. GRAPe-2A [6] has pipelines of digital signal processors to perform molecular dynamics (MD) simulations. MODEL [19] is an ASIC machine for MD simulation as well. Authors in Refs. [1] and [3] show the feasibility of using field programmable gate arrays (FPGAs) to implement large-scale application-specific computations.

As a part of an ongoing project [18], the LSRDP architecture is designed and implemented. Further, necessary tools are developed for extracting data flow graphs (DFGs) from the critical sections of the target scientific applications as well as configuration bit-streams for the LSRDP. To the best of authors’ knowledge, the LSRDP is the first high-performance computing system which is implemented using superconductivity circuits for accelerating scientific applications. Although, we have used the conventional design flow, the constraints originating from SFQ circuits and LSRDP’s unique architecture have been applied within the design procedure. For instance, using particular routing network composed of ORNs, lack of long connections over the PE rows and utilization of transfer units in the PEs as intermediate routing resources impose some specific limitations on the placement and routing procedures which necessitates customization of the conventional design flow.

Compiler support for the LSRDP architecture is an essential requirement during the design procedure and hardware utilization phase. In Ref. [4] a design methodology for their own coarse-grained reconfigurable data-path architecture has been proposed. The design methodology can only serve the certain applications and accelerate the loops of the attempted applications. Majority of previous efforts at compilation tools have focused on exploiting instruction-level parallelism (ILP) [2,7]. Mei et al. in Ref. [11] propose a modulo scheduling algorithm based on simulated annealing which begins with a random placement of operations on the FUs of a coarse-grained reconfigurable architecture. Modulo scheduling is a software pipelining technique that exposes parallelism by overlapping successive iterations of a loop [17]. The goal is to find a valid scheme for a loop such that the interval between successive iterations (initiation interval or II) is minimized.

A modulo graph embedding approach has been proposed in Ref. [14] in which the scheduling algorithm is reduced to placing operations of a loop body on a three-dimensional grid including FU array and the time slots of a modulo scheduled loop that form the third dimension.

Scheduling for the LSRDP differs from scheduling the CGRA due to the different hardware characteristics. CGRA is a generic template which usually uses register files and the degree of register file sharing varies from small, individual RF at each PE to multiple RFs each shared by a small number of PEs to a single central RF accessible by some or all nodes [14]. Unlike, the LSRDP does not use a RF but input/output streams buffers are used instead. Besides, various interconnection networks can be supposed for the CGRA such as connections between each PE and its nearest neighbors, buses connecting each PE to some other PEs in the same row or column, hierarchical connection schemes, etc. On the contrary, LSRDP comprises a certain interconnection topology for unidirectional connections from each row to the neighboring row. ORNs are responsible for providing connection resources form PE at a row to the PEs located in a certain distance at consecutive row. In addition, for connecting the PEs located in consecutive rows transfer units are employed as intermediate routing resources. As mentioned before, inherent latch function of SFQ circuits provides the architecture with pipeline facilities. Instead of storing intermediate data in RFs as available in CGRA, the intermediate data is latched at the PEs in each clock cycle. Therefore, the II-cycle in LSRDP architecture is equal to the LSRDP height (the number of LSRDP rows).

3. General architecture and specifications of the LSRDP

Generally, LSRDP is a pipelined architecture comprising a two-dimensional array of processing elements (PES) and operand
routing networks (ORNs) such that one PE can be connected through ORNs to a number of PEs in the next row. The LSRDP is implemented by SFQ circuits as a substitute to CMOS circuits. Fig. 1a displays the overall architecture of the proposed high-performance computer consisting of a GPP, the LSRDP as accelerator and memory elements.

Data flow graphs are pulled-out from critical segments of applications and configuration bit-streams are generated by using a dedicated tool. During execution of an application, configurations associated with the critical segments are loaded onto the LSRDP and executed in favor of achieving higher performance and lower power consumption. Since the cascaded PEs can generate a final result without temporally memorizing intermediate data, the number of memory load/store operations corresponding to spill codes can be reduced. Therefore, memory bandwidth required to gain a high-performance computation might decrease as well. Furthermore, as a loop-body mapped onto the PE array is executed in a pipeline fashion, LSRDP can provide a high computing throughput.

Within the design procedure except the basic properties of the LSRDP architecture, it is intended to determine the specifications of the architecture including following ones.

3.1. Input/output ports

I/O ports are located on top and bottom boundaries of the LSRDP as displayed in Fig. 1a. Limitation on the number of ports depends on available memory bandwidth, LSRDP operation frequency, width of data bus and the number of memory read/write channels.

3.2. LSRDP dimensions

Fig. 1a shows the height and width of LSRDP as the number of rows and columns, respectively.

3.3. Layout

Layout of the LSRDP indicates the type of functional units and their distribution (Fig. 1b). Three types of layout are examined for the LSRDP during the design procedure. In a normal layout (Layout-I), each FU can implement any operation including ADD/SUB and MUL. In Layout-II, every other PE implements only one of the ADD/SUB and MUL operations. In the third layout, every other row of the LSRDP implements one of the operations. In other words, all PEs in the first row only implement multipliers while in the second row adders are implemented solely.

3.4. PE types

Three types of PE architecture are examined for the LSRDP (Fig. 1c). Most suitable PE architecture is selected during the design procedure. The first PE architecture (PE I) includes a FU for implementing desired operation and a TU (transfer unit). As ORNs provide only routing resources between consecutive rows, TUs are utilized to connect two PEs locating on inconsecutive rows. It is possible to use a FU for implementing a transfer unit as well. In addition, each PE has three inputs (two inputs for FU and one for the TU) and two outputs (one from FU and another from TU). The second PE architecture (PE II) has one additional TU for increasing the flexibility of routing and it has 4-ins/3-outs. The third type of PE architecture (PE III) resembles the first one, though the difference is in extending capability of implementing two simultaneous TUs by the FU (totally three TUs). An additional mux should be used inside the PE to choose between FU’s output and the input. Various functionalities for the three PE types have been depicted in Fig. 1d.

3.5. Type and granularity of functional units

Each FU can implement basic 64-bit double-precision floating point operations, e.g., ADD, SUB and MUL. Control instructions (branches) and direct memory accesses via PEs are not supported.

3.6. Operand routing network (ORN)

PEs of each row are connected to a number of PEs in the next row through ORNs as routing resources. Fig. 1e shows the definition of connection length and the maximum connection length (MCL) on a piece of LSRDP architecture. It can be seen that the connection length of two PEs is the horizontal distance between them.

![Fig. 1. Overall architecture of the SFQ-LSRDP computer.](image-url)
Correspondingly, the MCL size is the maximum horizontal distance of two consequent PEs located in two subsequent rows. ORNs should provide all outputs of a PE with totally no_of_inputs \times (2 \times \text{MCL} + 1) connections to the PEs of consequent row. ORNs’ functionality is similar to a multiplexer however; ORNs are composed of cross-bar switches (CBs). Similar to other components of the LSRDP, CBs are also implemented by means of Josephson Junctions as the basic elements of the SFQ circuits [5].

A typical ORN structure locating between two rows of PEs is displayed in Fig. 1f. The crossbar-based ORN has a regular pipelined structure that does not limit the performance of the LSRDP and can be reconfigured on the fly. It can also be easily re-designed and expanded for any given complexity by adding a necessary number of extra rows of crossbars [5]. A sample chip implementation of a typical ORN is shown in Fig. 1g.

A unified ORN is implemented between every two rows rather than multiple ORNs. Each PE in one row can be connected to any of 2 \times \text{MCL} + 1 PEs in the consecutive row via dedicated ORNs. Three factors including the PEs architecture, MCL size and the LSRDP width are basic factors which highly impact the ORN structure and size. In Fig. 1f, PE I including three inputs and two outputs is employed. Assuming W as the LSRDP width, the ORN will totally consist of 2 \times W \times (4 \times \text{MCL}) CBs.

3.7. Internal memory

64-bit immediate registers are located in each PE in order to handle immediate operands (Fig. 1h).

3.8. Reconfiguration mechanism

LSRDP is an adaptable hardware that can be reconfigured within run-time using the bit-stream generated for the DFGs. Fig. 1h shows the architecture of a PE and how it can be reconfigured during the configuration phase. Apart initializing immediate registers, the multiplexers, PEs and ORN micro-routing network should also be programmed using the configuration bits. In order to configure each component, the configuration bit-stream is serially transferred to configuration registers.

4. LSRDP design procedure and the mapping tool

Compiler support for the LSRDP architecture is an essential requirement during the design procedure and hardware utilization phase. The extracted DFGs possess a large number of operations which necessitates a sophisticated mapping tool to locate DFGs onto the LSRDP and satisfy the architectural constraints as well. Fig. 2 shows the proposed tool chain which is similar to the conventional ones, however it has been customized for the LSRDP processor. The basic functionality of this flow is to generate configuration bit-stream and an executable code for the reconfigurable processor. Firstly, a hw/sw partitioning is performed on input application. Critical segments of the code are isolated and DFGs are extracted manually, however automatic generation is possible for some of applications (e.g. finite difference equations [15]). Applying the LSRDP architectural constraints, the DFGs are mapped onto the LSRDP through placing DFG nodes on the PE array, routing interconnection as well as assigning input/output nodes to the proper I/O ports. Limitations originated from the basic architecture of the LSRDP and from SFQ circuits (e.g. sparse routing resources, difficulties of using memory elements and register file for intermediate data, etc.) necessitate particular placement and routing algorithms.

Within mapping phase one main objective is to minimize the longest connection length such that the interconnections could be successfully routed. We will show the longest connection should be minimized for reducing the ORN size. Placement and routing procedures ought to be iterated until a valid map satisfying the LSRDP constraints are generated. Configurations’ bit-stream associated with each one of DFGs can be generated after completion of the mapping stage. An executable code including non-critical segments of the application code and a piece of code for LSRDP interfacing has to be generated. A part of compiler tools is customized to be utilized in the LSRDP design phase as shown in Fig. 2.

Determining following essential architectural specifications of the LSRDP is aimed during the design procedure:

- the total number of required PE in the PE array,
- suitable layout and PE architecture,
- the number of PEs in each row (width),
- the number of PE rows (height),
- the number of input/output ports,
- suitable ORN architecture with minimum number of CBs,
- the total number of ORNs,
- the size of ORNs including the numbers of inputs/outputs and CBs for each ORN which necessitates obtaining the MCL size as well.

Further, the main optimization objective is to minimize the number of resources, hence reducing the overall LSRDP area.

Entire design procedure is an iterative process of gathering statistics and analyzing them. We used a quantitative approach for designing the LSRDP architecture and determining its detailed architectural specifications. To determine each design parameter, DFGs should be mapped onto the LSRDP and the outcome is analyzed. The mapping process is performed without forcing any constraint except ones originated from the LSRDP basic properties, e.g., unidirectional data flow over the PE rows, availability of routing resources only between subsequent rows, etc. In the next stage, the results of mapping should be analyzed by the designer to decide an appropriate value for the intended parameter.

4.1. Benchmark applications

There are many applications which are the target of high-performance computing systems. An example is a molecular orbital calculation [13] which has a key role in theoretical prediction of various chemical properties. The amount of floating point operations for each calculation is between 122 and 1237 and the critical path length is from 13 to 76. To execute this application efficiently, PC cluster computers or vector type computers have been utilized so far [12].

Four various calculations are attempted as scientific benchmark applications including: one-dimensional heat (referred as Heat) and vibration equations (Vibration), two-dimensional Poisson equation (Poisson) [15], and recursion calculation part of electron repulsion integral (ERI) [13] as a quantum chemistry application. Calculations consist of only ADD, SUB, and MUL operations. Generally, extracting data flow graphs from applications can be performed manually or automatically by means of a sophisticated high-level profiling tool. In the former case, programmer needs to have a sufficient knowledge on the application and its detailed characteristics. We have implemented the above benchmark applications and then analyzed them to identify the critical segments and extract appropriate set of DFGs.

Eq. (1) is the one-dimension Heat equation which can be solved by finite difference method (FDM) as Eq. (2) in which B and D are...
constants. This finite difference equation shows that the next point during the time evolution process \((x_i, t_{j+1})\) is obtained by using current three points: \((x_i, t_{j+1}), (x_i, t_j)\) and \((x_i, t_{j-1})\). This equation can be applied to a one-dimensional \(N + 1\) points: \(x_{i,N/2} \sim x_{i,N/2}\) and it can be repeated \(M\) times to calculate from \(t_j\) to \(t_{j+M}\). A basic computation core for Eq. (1) can be shown as Fig. 3a, however employing such small units on a large-scale data-path accelerator is not likely cost-effective. By combining \(N\) DFGs over the space and \(M\) over the time directions, a larger DFG with \(N\) inputs and \(N \cdot 2^M\) outputs as well as \(4 \cdot (2N - 2^M) \cdot M/2\) operations is produced (Fig. 3b).

\[
\frac{\partial T(x, t)}{\partial t} = \frac{\partial^2 T(x, t)}{\partial x^2} \tag{1}
\]

\[
T(x_i, t_{j+1}) = D \times T(x_i, t_j) + B \times [T(x_{i-1}, t_j) + T(x_{i+1}, t_j)] \tag{2}
\]

A similar DFG generation procedure is applicable to the basic DFGs of vibration and Poisson equations. DFG generation for ERI is more complicated than other abovementioned applications, therefore, it is processed manually. Table 1 shows the generated DFGs and their specifications. The number of nodes represents the total number of DFG nodes including input/output nodes and the pure double-precision floating point operations. Further, the maximum fan-out in the two last columns indicates the maximum number of children for a DFG node.

4.2. Placement and routing

Mapping process consists of two sub-procedure, i.e., placement and routing. Throughout the mapping process, DFG nodes are placed on appropriate positions (PEs) over the LSRDP. This is similar to the well-known placement problem. Generally, minimizing the total connection length or the maximum connection length are main objectives, however in designing the LSRDP, the main goal is to minimize the maximum connection length (MCL) that directly impacts the ORN sizes and the LSRDP total area.

Our placement algorithm uses the conventional ASAP (as soon as possible) schedules which are representing a priority for placing DFG nodes on the PEs rather than the exact timing. Routing process establishes connections between the source and target PEs in the LSRDP by means of routing resources including ORNs and transfer

---

**Fig. 2.** A detailed outline of the proposed hw/sw compilation flow.

**Fig. 3.** (a) Data flow graph of basic heat equation (b) combined data flow graph of Heat equation.
units. As aforementioned, it is supposed that each PE can implement one or a couple of transfer units for passing data to consecutive rows. For each connection it is aimed to find a shortest path between the source and destination with the minimized MCL.

Similar to the edge-centric modulo scheduling algorithm [14] our algorithm is not confined to assigning PEs and a time slot to an operation (as in traditional schedulers), but it also takes routing of connections between operations into consideration. Placement and routing phases are integrated and positioning a node is committed while the routing procedure is completed without failure. Moreover, unlike most of previous algorithms which are fully or partially using simulated annealing or other non-deterministic algorithms ours is a deterministic one. The main difference is considering the constraints and specifications of the LSRDP architecture and customizing the proposed algorithm for the LSRDP architecture. As mentioned before, the LSRDP architecture development highly depend on limitations that the superconductivity circuits impose on it.

### 4.3. I/O nodes placement

Input/output nodes of the DFG should be assigned to appropriate input/output ports of the LSRDP on the boundaries. Between the first/last LSRDP rows and input/output ports, ORNs are available as routing resources. The main objective is to reduce connection length between input/output ports and PEs in the first/last row of the LSRDP. Firstly, input nodes are placed on the proper position so that the length of connections to their children are minimized. Then the integrated and positioning a node is committed while the routing procedure is completed without failure. Moreover, unlike most of previous algorithms which are fully or partially using simulated annealing or other non-deterministic algorithms ours is a deterministic one. The main difference is considering the constraints and specifications of the LSRDP architecture and customizing the proposed algorithm for the LSRDP architecture. As mentioned before, the LSRDP architecture development highly depend on limitations that the superconductivity circuits impose on it.

### 4.3.1. Proximity factor

For each pair of input nodes \( i \) and \( j \) the proximity factor is calculated as:

\[
p_{ij} = \sum_{k \in S_{ij}} \frac{1}{D_{ki}}
\]

while \( D_{ki} \) is the distance of common descendant node \( k \) to the input ports \( i \) and \( j \). The distance of a node to its related input port can be calculated as its ASAP level of execution. \( S_{ij} \) is the set of common descendants for the input nodes \( i \) and \( j \). Larger number of common descendants with smaller distance to the parents pair result in larger proximity factor and therefore, corresponding nodes should be placed in a closer distance to each other. Proximity factors for different pairs of the sample DFG have been calculated and shown in Fig. 5.

### 4.3.2. Placement algorithm

A heuristic algorithm is introduced below which employs the proximity factor for the node placement. Here are some definitions.

- \( P \) is the matrix of proximity factor for input node pairs.

\[
P = \begin{bmatrix} p_{1,1} & \cdots & p_{1,n} \\ \vdots & \ddots & \vdots \\ p_{n,1} & \cdots & p_{n,n} \end{bmatrix}
\]

- \( P_{ij} = \frac{1}{D_{ki}} \) if \( i = j \)

\[
P_{ij} = \begin{cases} \infty & \text{if } i = j \\ \sum_{k \in S_{ij}} \frac{1}{D_{ki}} & \text{if } i \neq j \end{cases}
\]

- \( L \) is the input ports array which stores the list of placed nodes such that the indexes indicate the location of corresponding input node.

- \( l \) and \( r \) denote the index of candidate locations in \( L \) for placing the under process input node.

\( C_{im} \) and \( C_{rm} \) represent the amount of proximity of an under process node \( m \) to the previously placed input nodes which have been located between \( l \) and \( r \) in array \( L \). The node \( m \) will be placed in location \( l \) if \( C_{im} \) is larger than \( C_{rm} \) otherwise, it will be located in location \( r \). \( C_{im} \) and \( C_{rm} \) are calculated as:

\[
C_{im} = \sum_{i=l+1}^{r-1} \left( p_{im} \times \frac{1}{|i-l|} \right)
\]

\[
C_{rm} = \sum_{i=l+1}^{r-1} \left( p_{im} \times \frac{1}{r-i} \right)
\]
In Eqs. (6) and (7), $p_{i,j}$ is the proximity factor of node $m$ and the already placed node $i$. The second term is the inverse of distance between candidate location ($l$ or $r$) and the location of node $i$. In this way, the amount of proximity to already placed nodes is examined and one of the candidate locations ($l$ or $r$) is chosen.

Placement alg.

1. Construct matrix $P$ including the proximity factors for each pair of input nodes ($n$ is the number of input nodes). Initialize $L = \Phi$.
2. Find node $m$ with the highest proximity factor from the first row of matrix $P(i=1)$ and place it in array $L$ so that $L[n/2] = m$.
3. Initialize $l$ and $r$ to $n/2 - 1$ and $n/2 + 1$, respectively.
4. Find the next node ($m$) with the highest proximity factor from the first row of matrix $P(i=1)$.
5. Calculate $C_{lm}$ and $C_{rm}$ using Eqs. (6) and (7).
6. if $C_{lm} > C_{rm}$:
   \[ l = |l + 1|, L[l] = m \text{ (node } m \text{ is placed in the location } l) \]
   else:
   \[ r = |r + 1|, L[r] = m \text{ (node } m \text{ is placed in the location } r) \]
7. if still there is any unplaced input node, go to step 4.

4.4. DFG placement

DFG nodes which implement the operations can be mapped by using a slightly modified proximity-factor based algorithm. Firstly, ASAP scheduling algorithm is used for calculating the execution order of each node. Obviously, direct children of the input nodes have the smallest ASAP level and the direct parents of output nodes have the largest levels of execution. In a descending order, groups of the nodes with similar levels of execution are subjected to the proximity-factor based placement algorithm.

Only a small difference exists between placing the input nodes and operational nodes. In the modified algorithm, placement is not limited to only one row of PEs but, the nodes can be distributed over the PEs array to be able to minimize the horizontal distance as much as possible. A modified connection length measurement is also defined in the next subsection that can result in smaller connection length. Finally, the output nodes are located on an array of output ports. Only the locations of parent nodes are taken into account when placing them so that the maximum horizontal connection length to their parents is being minimized. The proximity-factor based algorithm is not applicable for the output nodes which do not have any descendants.

4.5. Connection-length minimization

In the basic placement algorithm, connection length between any source and destination nodes is evaluated as the horizontal distance of two nodes (connection length = $h_s + h_d$: horizontal distance). For a couple of nodes locating in consecutive rows ($d_v = 1$), the only possible way for routing is through available ORN resources. On the other hand for two nodes placed in inconsecutive rows (vertical distance: $d_v > 1$), it is possible to use intermediate TUs for routing. In this way, the connection can be segmented to $d_v - 1$ connections between consecutive rows.

Considering above point, we propose an alternative measurement for the connection length as $h_{sm} = |d_h/d_v|$. In Fig. 6 (left-side) the two definitions of connection length have been displayed. In the right-side two connections (denoted by 1 and 2) can be seen so that $h_{sm} = 1$, while they have different vertical distances ($d_v = 3, d_v = 2$). For connection ‘1’, connection length would be 3 while the only possible way for routing from src to dest1 is via the ORN switches. On the contrary, for connection ‘2’, it is possible to use available intermediate TUs and break it into three segmented connections from src to dest2. This results in reducing connection length to one.

By using this new connection length measurement, the placement algorithm was modified such that the vertical connection length becomes effective in calculating the cost function. Fig. 7 shows an example of placing a node which has two parents ($P_1$...
and P2) already placed on the PE array. Obviously, due to the routing resource constraints (particularly of being unidirectional) descendant nodes can be placed on the PEs of their parents’ succeeding rows (indicated as candidate rows in Fig. 7).

The connection length has been calculated based on two above-mentioned measurements for each candidate PE. By using \( l_h \) for connection length measurement, PE2 is the best choice which gives the minimum MCL equal to two. In the second candidate row, choosing PE8 gives the same result, however PE2 is preferred due to smaller vertical distance. On the other hand, connection length measurement based on \( l_h \) results in choosing PE2 which gives the least value of MCL equal to one in the first row. Although PE6, PE7 and PE8 in the second row give the same result, but due to smaller vertical distance, PE2 is chosen eventually.

5. LSRDP design space exploration

Design of the LSRDP architecture entails a multitude of design parameters. Variety of design parameters indicates the high complexity of the design procedure and proves the requirements for a methodological approach. A major challenge within the design procedure is to find the right balance between the different quality requirements that a system has to meet. One approach to resolve it is design space exploration (DSE) which is the process of analyzing several “functionally equivalent” implementation alternatives to identify an optimal solution. Considering different design specifications too many number of design alternatives for a task might be generated.

Evidences from the experiments show that aiming the MCL reduction does not necessarily result in the overall area minimization. Obviously, the LSRDP area is the summation of PE arrays’ and the ORN networks’ areas. According to Fig. 8, the structure and area of an ORN are influenced by the LSRDP width, the PE structure and the MCL size. Therefore, ORN area is differently calculated for three PE types. The number of ORNs is equal to \( H + 1 \) and the total area is calculated as Eq. (8).

\[
\text{Area}({\text{ORNs}}) = \omega_2 \times (\omega_3 \times \text{MCL} + \omega_4) \times W \times (H + 1) \times \text{Area(CB)}
\]  

where coefficients \( \omega_2, \omega_3 \) and \( \omega_4 \) are obtained from Table 2 depending on the designated layout type and PE architecture.

The area of a PE is calculated differently for three PE architectures as well. It is very important to notice that under the SFQ implementation, a 64-bit double precision multiplier consumes the same amount of the Josephson junctions that an adder or subtractor consumes. Further, the area of a transfer unit as well as a multiplexer is about 10% of the area of an arithmetic operation of the same bit-width. Therefore, in case of using layout I and PE I so that each PE implements both multiplication and addition operations plus a transfer unit, the area of a PE would be equal to \( 2.1 \times \text{Area(ADD)} \) (Fig. 8a). For the layout II in which only one adder or multipliers is used, the area of a typical PE type I is equal to \( 1.1 \times \text{Area(ADD)} \). In Fig. 8b and c area calculation for PE II and III have been depicted, respectively. In PE III, the FU can implement two TUs, hence a multiplexer is required for selecting an output of the FU and an input. Consequently, the area of PE array in the LSRDP architecture can be calculated as Eq. (9).

\[
\text{Area(PEArray)} = \omega_1 \times W \times H \times \text{Area(ADD)}
\]  

different values of \( \omega_1 \) for three PE types have been depicted in Table 2. Consequently, the total area of the LSRDP which is the summation of two above terms in Eqs. (8) and (9) can be calculated as the following equation.

\[
\text{Area(LSRDP)} = \omega_1 \times W \times H \times \text{Area(ADD)} + \omega_2 \times (\omega_3 \times \text{MCL} + \omega_4) \times W \times (H + 1) \times \text{Area(CB)}
\]  

The LSRDP design space is explored for various MCL values, three PE architectures and three layouts to obtain the minimum LSRDP area. The LSRDP total area in each design point is calculated using the above equations and the design point giving minimum overall area is selected among the evaluated ones.

6. Experimental results

A tool chain mainly comprising the mapping tool was developed in accordance with the proposed ideas and it was utilized within the design procedure of the LSRDP architecture as well. During the design process, first the DFGs listed in are extracted from the target computation-intensive codes manually. Afterward, they are mapped onto an LSRDP architecture so that can meet the primary constraints of the architecture. In the initial phase, there is unlimited number of functional and routing resources although, the LSRDP has some basic constraints, e.g., limitation of the interconnection resources. In the next phase, outcome of the mapping process is analyzed to decide the LSRDP architectural specifications as enumerated in Section 3.

The number of input and output ports are calculated in the first step of the design process which are equal to 19 and 12, respectively. This calculation is performed considering the available memory bandwidth, LSRDP operation frequency, width of data bus and the number of memory read/write channels.

Firstly, it is tried to obtain the minimum MCL size in order to optimize the overall ORNs’ area. Table 3 depicts the results. The target LSRDP layout is type II and three PE architectures have been attempted. The proximity-factor based input nodes placement was attempted and \( l_h \) was utilized for the connection length measurement (Table 1). Further, the overall area of LSRDP has been reported in terms of the number of Josephson junctions (JJs). It is observed from the implementations that each CB needs around 550 JJs and each FU, i.e., ADD/SUB or MUL requires around 40 KJJs.
Observations from the experiments indicate that the smaller MCL size might reduce the ORN area but it does not necessarily result in the optimized overall area. Fig. 9 indicates evidences from two applications (i.e. Vibration and Poisson) for this finding. According to the graphs, ORN area slightly rises when the MCL size increments, even if the LSRDP dimensions increase. On the contrary, area of PE-array (as well as LSRDP dimensions including width and height) has a reduction when the MCL increases up to MCLopt. There is no change in the area of PE-array when MCL becomes larger than MCLopt. Variations of the overall area and PE-array's area are alike up to MCLopt, however it rises for larger MCL values due to the increase in ORNs area. This motivated us to explore the design space for LSRDP architecture and find the optimal MCL size and LSRDP dimensions so that minimum area for the LSRDP can be obtained.

Three PE structures as well as the three LSRDP layouts (Fig. 8) were examined across the design space exploration. Corresponding to the PE architectures, three ORN structures were evaluated. As shown in Table 4, due to simpler architecture of PE I which provides up to two TUs in each PE for three attempted layout types the MCL size is larger compared with the two other PE structures. Also, a large number of PEs are consumed mostly for the interconnections, hence PE I results in larger overall area due to less flexibility. The smallest MCL size is obtained when PE II is employed, however the overall area would be better than that of PE III when having layout I as underlying structure. The smallest overall area is achievable once PE III is employed in layout II.

Consequently, the LSRDP final specifications can be derived from Table 4 in which the MCL size, width and height are recognized as 4, 22 and 14, respectively. Also as aforementioned, the layout II is the most suitable one and PE architecture II can provide reasonably sufficient routing resources and routing flexibility to keep the overall area the smallest.

A comparison between the initial results and the outcomes of the design space exploration has been made in Table 5 to demonstrate the efficiency of using design space exploration for the LSRDP architecture. This comparison is only for layout II, because it represents the most appropriate underlying structure for the LSRDP. For the three PE structures, DSE obtains smaller area with

<table>
<thead>
<tr>
<th># of inp/out</th>
<th>( \alpha_1 )</th>
<th>( \alpha_2 )</th>
<th>( \alpha_3 )</th>
<th>( \alpha_4 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>Layout-I</td>
<td>2.1</td>
<td>1.1</td>
<td>1.1</td>
<td>1.5</td>
</tr>
<tr>
<td>Layout-II</td>
<td>2.2</td>
<td>1.2</td>
<td>1.2</td>
<td>2</td>
</tr>
<tr>
<td>Layout-III</td>
<td>2.2</td>
<td>1.2</td>
<td>1.2</td>
<td>1.5</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>PE arch.</th>
<th>Width</th>
<th>Height</th>
<th>MCL</th>
<th>ORNs area (KJJ)</th>
<th>PE-array area (KJJ)</th>
<th>Overall area (KJJ)</th>
</tr>
</thead>
<tbody>
<tr>
<td>PE I</td>
<td>27</td>
<td>17</td>
<td>17</td>
<td>27,115</td>
<td>20,196</td>
<td>47,311</td>
</tr>
<tr>
<td>PE II</td>
<td>20</td>
<td>17</td>
<td>3</td>
<td>8787</td>
<td>16,320</td>
<td>24,196</td>
</tr>
<tr>
<td>PE III</td>
<td>20</td>
<td>17</td>
<td>6</td>
<td>9846</td>
<td>16,320</td>
<td>26,166</td>
</tr>
</tbody>
</table>
the amount of improvements as 54.6%, 15.5% and 16.7%, respectively.

Currently, fabricating only small scale LSRDP architecture is possible however, implementing large-scale accelerator using SFQ circuits is our main target. We have obtained preliminary performance evaluation results through a simulation for two applications, i.e., Heat and Vibration. Experiments show by using the LSRDP machine including a 3.2 GHz out-of-order base processor and operating frequency of 80 GHz for the LSRDP, remarkable performance values, namely 210 Gflops and 104.9 Gflops are achievable, respectively.

7. Conclusion

We discussed about developing a high-performance low-power computer which is composed of a GPP and an accelerator as the main components. The proposed design procedure and the tool chain were explained as well. The target accelerator is a large-scale reconfigurable data-path computing unit with a large matrix of PEs implemented by SFQ circuits. That is suitable for executing massive computational-intensive scientific applications. A mapping routine is a major part of the developed tool chain in which the main goal is to optimize the overall LSRDP area. The tool chain has been employed to examine various PE architectures and layouts for the LSRDP though exploring the design space. Efficiency of the utilized strategy has been proved via the experimental results which prove noticeable improvements in LSRDP area compared with the initial results. According to the observations from simulation results, LSRDP is promising to achieve noticeable performance values in the range of hundreds of Gflops.

Acknowledgments

This research was supported in part by Core Research for Evolutional Science and Technology (CREST) of Japan Science and Technology Corporation (JST).

References


Farhad Mehdipour received the B.Sc. degree from Sharif University of Technology in 1996, and the M.Sc. and Ph.D. degrees in Computer Systems Architectures from the Amirkabir University of Technology in 1999 and 2006, respectively. He was a visiting researcher to the Kyushu University during November 2005 to June 2006. From December 2006 to July 2010 he was working as a researcher in the Department of Informatics at Kyushu University and he is currently an associate professor in the E-JUST center at Kyushu University. He is a member of IEEE and his research interests include reconfigurable processors and high-performance computing systems.

Hiroaki Honda received the Ph.D. degree in the graduate school of science of Hokkaido university, Japan in 2000. Currently, he is a researcher of Institute of Systems, Information Technologies and Nanotechnologies. His research interests are quantum chemistry, computational chemistry, and high-performance computing. He is a member of the Chemical Society of Japan, American Chemical Society, and Association for Computing Machinery.

Koji Inoue received the B.E. and M.E. degrees in computer science from Kyushu Institute of Technology, Japan in 1994 and 1996, respectively. He received the Ph.D. degree in Department of Computer Science and Communication Engineering, Graduate School of Information Science and Electrical Engineering, Kyushu University, Japan in 2000. In 1999, he joined Halo LSI Design and Technology, Inc., NY, as a circuit designer. He is currently an associate professor of the Department of Informatics, Kyushu University. His research interests include power-aware computing, high-performance computing, dependable processor architecture, and secure computer systems. He is a member of the ACM, the IEEE, the IEEE Computer Society, the IEICE, and the IPSJ (Information Processing Society of Japan).

Hiroshi Kataoka received the B.E. and M.E. degrees in computer science from Kyushu University, Japan in 2008 and 2010, respectively. He is working toward Ph.D. in Graduate School of Information Science and Electrical Engineering, Kyushu University, Japan.

Kazuaki Murakami received the B.E., M.E., and Ph.D. degrees in computer science and engineering from Kyoto University in 1982, 1984, and 1994, respectively. From 1984 to 1987, he worked for the Fujitsu Limited, where he was a Computer Architect of the mainframe computers. In 1987, he joined the Department of Information Systems of Kyushu University, Japan. He is currently a Professor of the Department of Informatics and a member of the ACM, the IEEE, the IEEE Computer Society, the IPSJ, and the JSIAM.