Mobile Information Systems 6 (2010) 65–83
DOI 10.3233/MIS-2010-0093
IOS Press

Balanced bipartite graph based register allocation for network processors in mobile and wireless networks

Feilong Tang[a,b], Ilsun You[c,*], Minyi Guo[a], Song Guo[b] and Long Zheng[b]
[a]*Department of Computer Science and Engineering, Shanghai Jiao Tong University, Shanghai 200240, China*
[b]*School of Computer Science and Engineering, The University of Aizu, Fukushima 965-8580, Japan*
[c]*School of Information Science, Korean Bible University, 16 Danghyun 2-gil, Nowon-gu, Seoul, South Kroea*

Abstract. Mobile and wireless networks are the integrant infrastructure of mobile and pervasive computing that aims at providing transparent and preferred information and services for people anytime anywhere. In such environments, end-to-end network bandwidth is crucial to improve user's transparent experience when providing on-demand services such as mobile video playing. As a result, powerful computing power is required for networked nodes, especially for routers. General-purpose processors cannot meet such requirements due to their limited processing ability, and poor programmability and scalability. Intel's network processor IXP is specially designed for fast packet processing to achieve a broad bandwidth. IXP provides a large number of registers to reduce the number of memory accesses. Registers in an IXP are physically partitioned as two banks so that two source operands in an instruction have to come from the two banks respectively, which makes the IXP register allocation tricky and different from conventional ones. In this paper, we investigate an approach for efficiently generating balanced bipartite graph and register allocation algorithms for the dual-bank register allocation in IXPs. The paper presents a graph uniform 2-way partition algorithm (FPT), which provides an optimal solution to the graph partition, and a heuristic algorithm for generating balanced bipartite graph. Finally, we design a framework for IXP register allocation. Experimental results demonstrate the framework and the algorithms are efficient in register allocation for IXP network processors.

Keywords: Register allocation, network processor, bipartite graph, mobile and wireless network, register bank

1. Introduction

Mobile and pervasive computing (MPC) is an exciting new paradigm that provides "disappeared" services for people even while moving around, where mobile and wireless networks with powerful computing and communication abilities are the integrant infrastructure to make this vision a reality. More and more applications, for example, mobile video playing, require a high end-to-end network bandwidth to ensure the specified QoS requirements in such environments [2,30].

Powerful computation ability is requisite to high bandwidth in order to meet on-demand basic network services, e.g. packet classification and forwarding, as well as intrusion detection and firewalls. Routers are the core components of high-speed networks [10,14,23]. In general, packets are received and sent

*Corresponding author. Tel.: +82 2 950 5494; Fax: +82 2 950 5411; E-mail: ilsunu@gmail.com.

though input and output buffer which reside in on-chip memory that is characterized by large capacity and long access latency. A router should process packets at line rates of high speed, and at the same time be sufficiently programmable and flexible to support current and potential services. Traditional ASIC processors cannot meet such requirements because of their limited processing ability, and poor programmability and scalability.

Network processors (NP) emerge to address such high-speed challenges that are beyond the ability of general-purpose processors, while providing flexibility and programmability. It is a new breed of packet forwarding engine that is designed to meet the simultaneous demands of high speed and great flexibility of today's network equipments [35]. The goals are to provide the performance of traditional ASICs and the programmability of general-purpose processors [16].

Keeping up with external line rate without losing packets makes it imperative that all latencies be kept to a minimum. For this purpose, network processors, for example, Intel IXP [4,25], in general contain a large number of registers to decrease the number of memory accesses [26]. In such a circumstance, efficient register allocation plays an extremely important role for the network applications to achieve high throughput.

Register allocation is the process of multiplexing a large number of target program variables onto a small number of CPU registers. The goal is to keep as many operands as possible in registers to maximize the execution speed of application programs. The register file of Intel' IXP is built using two-ported register files: one read port and one write port. The area efficiency of two-ported registers relative to multiport registers is important in allowing the large number of registers to fit in the allocated silicon area. On the other hand, the use of two-port registers places some restrictions on which combinations of registers source operands can use legally for an instruction [21]. These restrictions make the IXP register allocator tricky and different from conventional ones. Two main differences are band conflict and bank imbalance.

This paper focuses on how to effectively allocate registers for IXP network processors to reduce the bank conflict and balance the bank loads as much as possible, based on the balanced bipartite graph. Our motivation is to increase routing speed through allocating tasks on registers of the IXP-based network process efficiently to improve the performance of mobile and wireless networks.

The rest of the paper is organized as follows. In the next section, we review related work carefully. Section 3 briefly describes the register file and register allocation problems on two banks based IXP. In Section 4, we present a fixed-parameter tractable algorithm (FPT) for uniform graph bipartite problem and a heuristic algorithm for balanced bipartite graph. The algorithms will be used in the register allocation framework presented in Section 5. Experiments and evaluation are reported in Section 6. Section 7 concludes our paper with the discussion on the future work.

2. Related work

Many significant results on register allocation have been reported. In this section, we carefully review related work on register allocation algorithm and architecture, as well as related proposals for IXP network processors.

Register allocation algorithms. Most schemes on the register allocation used the graph coloring algorithm, which was proposed by Chaitin [9] in 1982. In the graph coloring algorithm, a graph vertex represents a variable that will be assigned a register; a line between two vertexes means a conflict between the two variables so that the two variables can not be assigned to the same register. Such a graph is called the conflict graph. The graph coloring algorithm converts the register allocation into the graph coloring

problem and becomes the base of register allocation. Briggs et al. [24] improved Chaitin's work by an optimistic graph coloring algorithm. The basic idea is to put the conflicted variable to a temp stack when the conflict graph cannot be simplified. Optimistically, the conflict may be solved a little later. Chow and Hennessy used a frequency-sensitive heuristic graph coloring algorithm, which assigns the priority based on the usage frequency. This method is very effective to programs with loops. Lueh [11] proposed a fusion-based register allocation approach, assigning registers in units of basic blocks, where potential conflicts are limited in individual basic blocks. Appel et al. [3] divided the register allocation into two sub-problems: optimization of the insert of overflowed codes and optimization of the register merging. They solved the first sub-problem using integer linear programming (ILP) to achieve the optimal scheme of register allocation.

Zhuang' proposal [33] is the most relevant to our work. They proposed three heuristic approaches, which differ in the order of register allocation (RA) and bank assignment. The first one is Pre-RA Bank Allocation Approach, simplified Pre-RA. This approach assigns banks before registers. It breaks odd cycles through live range splitting rather than deleting edges to make the graph bipartite. An algorithm is invoked to near-balance the vertexes into two banks. The well-know RA algorithm [9] is used to allocate registers for each bank in Pre-RA. However, it suffers from difficulty to judge physical register pressure in the two banks before RA and often results in imbalance and large spilling cost. The second one is Post-RA Bank allocation Approach, simplified Post-RA. This approach firstly allocates registers with conventional RA algorithms. Post-RA is similar to the approach that IXP products often employ. The rest parts are almost the same as Pre-RA. Finally, the third approach is the mixed register allocation. Their experimental results show that Post-RA has better runtime performance but inserts the more instructions to resolve bank conflicts and balance bank load. Nevertheless, it splits vertexes regardless of how it will affect the rest of the graph and incurs unnecessary move instructions between banks. Unfortunately, these approaches only considered the register allocation for general processors, without involving the features of registers in IXP. Moreover, Monreal et al. [32] presented a novel physical register management scheme that allows for a late allocation (at the end of execution) of registers to reduce the time accessing to register files, which significantly saves the number of registers and shortens the access time to register files. The approach used the virtual-physical registers, together with an on-demand register allocation policy and a stealing mechanism that prevents older instruction from being delayed by younger ones. The main idea is to allocate physical registers at the end of the execution stage, rather than at decode time.

Register allocation architectures. Jang et al. [31] studies banked register file under VLIW architecture, where register banks are partitioned such that one bank is associated with each functional unit or by associating a cluster of functional units with each bank. Abella and Gonz ãlez [13] proposed an adaptive micro-architecture that achieves significant dynamic and static power savings in the register file, at the expense of a very small performance loss. The proposed mechanism dynamically limits the number of in-flight instructions in order to save dynamic and static power in register files or rename buffers, and reduces register pressure, based on monitoring how much time instructions spend in both the issue queue and the reorder buffer and limit their occupancy, and taking resizing decisions based on these observations. Wagner et al. [20] proposes a special register allocation technique, where the register allocator maintains two sets of virtual registers: one for scalar values and one for register arrays. All virtual registers are indexed by a unique number. Any element of a register array can be accessed in two different ways: first by direct addressing or indirectly by the use of a bit-packet pointer. In case of insufficient physical registers using indirect access, spill code is generated for all virtual registers within a register array; otherwise, only the particular virtual register is spilled. Park et al. [19] studies

dual bank allocation problem. Nevertheless, they don't have the bank constraint like ours. Collins et al. [26] proposes a dynamic register allocation algorithm on IXP, but it requires modification to the hardware. Very Wide Register [29] is an asymmetric register file architecture, which has single ported cells and asymmetric interfaces to the memory and to the datapath, for low power embedded processors. The basic components in this organization are the interface to the memory, single ported cells and the interface to the datapath. The interface of this foreground memory organization is asymmetric: wide towards the memory and narrower towards the datapath. This foreground memory is similar to a register file which is incorporated in the datapath pipeline cycles. The wide interface enables to exploit the locality of access of applications through wide loads from the memory to the foreground memories (registers). At the same time the datapath is able to access words of a smaller width for the actual computations. ParShield [28] is a register architecture that provides cost-effective protection for register files against soft errors, which selectively protects a subset of the registers by generating, storing, and checking the ECCs of only the most vulnerable registers while they contain useful data. ParShield also adds a parity bit for all the registers and re-uses the ECC circuitry for parity generation and checking as well. In particular, ParShield has no performance impact and little area requirements. Kolson et al. [6] proposed a register allocation architecture to loop in multiple register files, where the available registers have been partitioned into multiple banks. A distinguishing characteristic of the approach is that the register assignment may span multiple iterations of the original loop. Hiser et al. [18] proposed a register assignment method for code generation for instruction-level parallelism (ILP) architectures with partitioned register banks, through the register component graph that abstracts away machine specifics within a single representation, especially for instruction-level parallelism architectures. The approach can produce good code, which increases the parallelism of operations, by separating partitioning from scheduling and register assignment.

Register allocation for IXP network processors. There also have been reports on register allocation for Intel IXP network processors. Shangri-La compiler [22] used a C-like high-level programming language developed by Intel, and provides the functions of optimal register allocation. George et al. [17] researched the single thread-based register allocation for IXP. They took many features of IXP into account, however, did not consider the important feature of IXP—multiple threads. Zhuang et al. [34] studies register allocation across threads on IXP and doesn't deal with the bank conflicts problem. Zhou [8] proposed a framework for allocation of registers of IXP, improving the work in [33] to some extent.

3. The IXP register allocation problems

The General Purpose Registers (GPRs) in an IXP network processor are physically split into two banks: Bank A and Bank B. The dual banks cause *bank conflict* and *bank imbalance* problems that have to been considered in register allocation for IXPs.

3.1. Bank conflict

For each instruction with two GPR source operands, one must come from Bank A and another from Bank B. The destination is not constrained. Certain allocation of GPR source operands may cause a conflict. For example, consider the three instructions sequence in Fig. 1(a). The first instruction above requires that A and B must be on different banks. The second instruction requires that A and C must also be on different banks. The above two allocations constrain that B and C must be allocated to the same

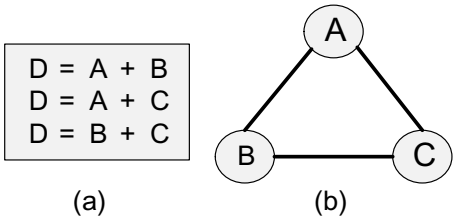

Fig. 1. Bank Conflict in IXP.

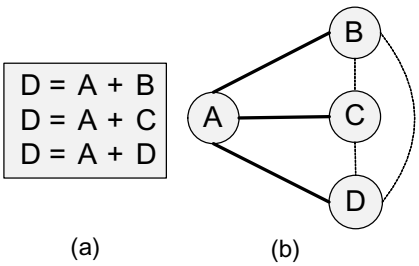

Fig. 2. Bank Imbalance in IXP.

bank since they are both in a bank different from that of A. However, the third instruction conflicts with this since it requires B and C to be allocated on different banks. Two variables are said to *conflict* with each other if they are specified in the same instruction as source operands.

These allocation problems can be understood and solved by some simple graphical analysis. An example of such a graph, called bank allocation graph (BAG) [12], is shown in the Fig. 1(b), corresponding to instruction sequence in Fig. 1(a). The edges in BAG indicate that the variables conflict with each other. These edges are called *conflict edges*. Whenever a cycle is formed with an odd number of edges, dual bank allocation is impossible [12], like in Fig. 1(a). It means that we have to resolve some bank conflicts. If the BAG is *odd-cycle-free*, we say it's *conflict-free.*

A simple solution provided in the IXP assembler is to inserts an instruction to copy one of the registers to a temporary register and replaces the register reference with a reference to the temporary register. The assembler attempts to minimize the number of inserted instructions, but it is error-prone and ineffective. A compiler solution is more efficient.

3.2. Bank imbalance

Another issue of register allocation in IXP is how to balance register workload in the two banks. The graph in Fig. 2(b) is the conflict graph of code segment in Fig. 2(a). The bold lines represent a conflict edge, while the normal solid lines represent that the two variables it connects merely interference with each other but do not conflict with each other. The BAG consists of the bold lines and the vertices incident with them. Obviously, BAG is a subgraph of the IG.

Assume there be two registers in each bank and B, C and D interfere with each other. The instruction sequence in Fig. 2(a) requires A in one bank and B, C and D in another. Nevertheless, the clique formed by B, C and D is not 2-colorable. So they cannot be allocated in the same bank with only 2 registers. This may make the assembler to issue a "*no enough registers*" error message although there may be a register available in the opposite bank. To deal with this, a compiler solution is preferred as well.

4. Fixed-parameter uniform 2-way graph partition

4.1. Bipartite graph and balanced bipartite graph

A bipartite graph is a basic conception in various register allocation algorithms. A bipartite graph is described as follows.

Bipartite Graph:

Given an undirected graph $G = (V, E)$. G is a bipartite graph if the set of vertexes E can be cut into two disjoint subsets A and B, and two vertexes, connected by each edge, locate at the subsets A and B respectively.

Bipartite graph prevents register allocation from bank conflicts because conflicted variables are assigned in different banks. To balance tasks in different banks, however, register allocation for IXP needs to find the balanced bipartite graph, which minimizes the tasks' difference between two subsets of a graph. A balanced bipartite graph is described below.

Balanced bipartite graph:

Given a partite graph $G = (V, E)$ with two subset G1$\subseteq$G and G2$\subseteq$G such that G1$\cup$G2=G. Let $|V1|$ and $|V2|$ be the numbers of vertexes in G1 and G2, respectively. We call G is a balanced bipartite graph if and only if $||V1| - |V2|| \leqslant 1$.

4.2. Fixed-parameter tractable graph bipartition

Many problems in compiler technique are NP-complete, e.g. register allocation via graph coloring. Approximate solutions are given by heuristic algorithms. Known algorithms for these problems to achieve an *optimal* solution require exponential time related to the size of input. Existence of efficient and exact algorithms is considered unlikely. However, this view has been challenged by *parameterized complexity* [27]. Some problems can be solved by algorithms that are exponential only in the size of a fixed parameter while polynomial to the input size. Such algorithms are called fixed-parameter tractable algorithms. These algorithms are of high practical interest for they are able to provide *optimal* solutions within small runtime for small parameter values.

The classic problem, *graph bipartition*, is NP-complete. Guo et al. [15] in 2005 provided an algorithm solving the graph bipartition problem by deleting vertices edge deletion version that runs in $O(2^k \cdot m^2)$ time, where m, n and k is the number of edges, vertices and number of edges to delete respectively. It means that *graph bipartite* problem is FPT. Their result show that the algorithm finishes within minutes for some graph instances with $n = 300$, $m = 1500$ and $k = 40$. The graph bipartition problem can be formally described as below:

Graph Bipartition:

Given an undirected graph $G = (V, E)$ and a nonnegative integer k, find a subset $C \subseteq E$ of edges (or vertexes) with $|C| \leqslant k$ such that after removal of C, the resulting subgraph $G' \subseteq G$ is bipartite. C is called *cut set*.

The Guo's algorithm gives a C with $|C| \leqslant k$ or proves that such a sub set C does not exist. The idea is to use a routine to compress the know solution iteratively. Given a size-($k+$ 1) solution, the routine either computes a size-k solution or proves that there is no size-k solution.

We present a graph uniform 2-way partition algorithm by improving Guo's algorithm so that it can deal with a weighted graph and give a minimum weighted cut set, which is the total weights of edges in it is minimal, with size not more than k. The resulting graph is called *maximal bipartite graph*. We

call the algorithm as FPT because it is a fixed-parameter tractable graph bipartition, where the key idea is how to maximize the uniform 2-way graph partition.

Maximal uniform 2-way graph partition:

Given a graph $G = (V, E)$ with weights on its edges and the number of vertexes, partition the vertexes of G into two subsets of equal size (or the difference is within 1) that the sum of weights of edges connecting different subset is maximal. We call the process for finding a balanced bipartite graph as maximum uniform 2-way graph partition.

It is equivalent to make the graph bipartite into two equal size with minimal cost since the remaining weights of edges will be maximal. The proof is trivial and we ignore it here. We just show how to get *uniform 2-way partite graph* from a *maximal bipartite graph*.

As for a bipartite graph, it has the property that its vertexes can be divided into 2 disjoint subsets such that all its edges have one endpoint in one set and another endpoint in the other set. Assume the maximum bipartite graph we get in last subsection is divided into such 2 disjoint subset, say *LEFT* = $\{L_1, L_2, \ldots, L_i\}$ and *RIGHT* = $\{R_1, R_2, \ldots, R_j\}$ as shown in Fig. 3. The bold lines represent edges in E', which called *connecting set*, and the normal solid lines represent edges in C. Note that C is not part of the maximum bipartite graph. Then we have the following definitions and claim:

Definitions:

A *flip* is a movement of a vertex from *LEFT* to *RIGHT* or vice versa.

$W\{L_x, \{L^*\}\}$ is the sum of weights of edges between L_x and all the nodes in set *LEFT*.

$W\{L_x, \{R^*\}\}$, $W\{R_x, \{L^*\}\}$ and $W\{R_x, \{R^*\}\}$ is defined similarly.

$W\{E\}$ is the sum of weights of edges in set E.

Claim: $\forall \quad L_x \in$ *LEFT* and $R_x \in$ *RIGHT*, we have

$W\{L_x, \{R^*\}\} - W\{L_x, \{L^*\}\} \geqslant 0$ and

$W\{R_x, \{L^*\}\} - W\{R_x, \{R^*\}\} \geqslant 0$.

Proof: Suppose for an L_x in *Left*, $W\{L_x, \{R^*\}\} - W\{L_x, \{L^*\}\} < 0$, then if we flip L_x to *Right*, the edges $\{L_x, \{R^*\}\}$ become part of the new cut set C' and $\{L_x, \{L^*\}\}$ become part of the connecting set, $E' = E - C'$. We have $W\{C'\} = W\{C\} + W\{L_x, \{R^*\}\} - W\{L_x, \{L^*\}\} < W\{C\}$. It contradicts with the fact $W\{C\}$ is the minimum cut set. $W\{R_x, \{L^*\}\} - W\{R_x, \{R^*\}\} \geqslant 0$ can be proved similarly.

This means that any flip will not decrease the weight of the new cut set.

Assume $|\,LEFT\,| \leqslant |\,RIGHT\,|$. To balance the two sets, i.e. to make $||\,RIGHT\,| - |\,LEFT\,||$ minimal, we have to flip $q = (|\,RIGHT\,| - |\,LEFT\,|)/2$ vertexes from *RIGHT* to *LEFT*.

We can only choose the q vertexes $\{N_1, N_2, \ldots, N_q\}$ such that

$\{W\{R_x, \{L^*\}\} - W\{R_x, \{R^*\}\} < W\{R_y, \{L^*\}\} - W\{R_y, \{R^*\}\} \mid x \in [1, q], y \in [q+1, |RIGHT|]\}$ because every flip will no decrease the weight of new cut set. This approach assures the weight of new cut set to be minimal. This operation takes $O(q) < O(|V|)$ time.

4.3. A heuristic algorithm for balanced bipartite graph

The FPT algorithm presented in the last subsection can get optimal solutions, however, it will spend much more time if values of input parameters become large because the uniform 2-way graph partition is a NP-complete problem. So, we also design a heuristic algorithm for efficiently generating bipartite graph by modifying the K-way balanced graph algorithm proposed by Lee [5]. The key idea is to how to convert the balanced bipartite graph into the maximal cut set, which can be reduced to balanced bipartite graph with ease. We formally define the maximal cut set as follows.

Maximal cut set:

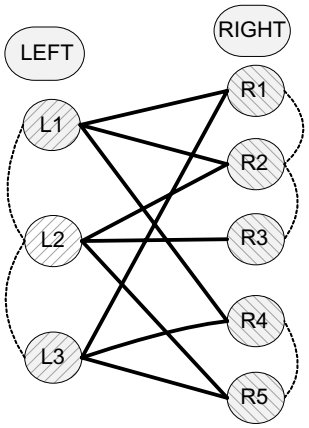

Fig. 3. A bipartite graph.

Given a undirected graph $G = (V, E)$ and *weights of all edges. We partition the set of* vertexes into two disjoint subsets A and B to maximize the sum of weights of edges such that two vertexes of each edge are in different subset, i.e.,

Maximal cut set $C = \{E_{ij} | \max \sum_{i \in A, j \in B} w_{ij}\}$, where W_{ij} is the weight of the edge between vertexes i and j.

The heuristic algorithm uses the conception gain. Let a graph G be partitioned into two subsets S_1 and S_2, and W(V_i, V_j) be the weight of the edge between vertexes V_i and V_j. The gain that vertex V_i moves from S_1 to S_2 is formally defined as:

$$g(v, S_2) = \sum_{v_i \in S_1} W(v, v_i) - \sum_{v_j \in S_2} W(v, v_j), \forall v \in S_1 \tag{1}$$

The heuristic algorithm is illustrated in Fig. 4. At the beginning, states of all vertexes in S_1, and S_2 are empty. The algorithm runs in a number of passes. In each pass, we take the result of last pass as the input, and move a vertex to S_2 such that the movement will increase the cut set. The selected candidate is the vertex with the maximal gain. A pass finishes if all vertexes are tried. Then, we take the result with the maximal cut set as the input of the next pass. The algorithm repeats until no new gain can get. Whenever a vertex V_r is moved, we need to recalculate the gains of all non-moved vertexes. The time complexity of each pass is $O(2|V|^2)$.

$$\begin{aligned} g(v_a, S_2) &= g(v_a, S_2) - 2W(v_a, v_r), \forall v_a \in S_1 \\ g(v_b, S_1) &= g(v_b, S_1) + 2W(v, v_r), \forall v_b \in S_2 \end{aligned} \tag{2}$$

4.4. Graph separation

At this point, we get a maximal uniform 2-way graph partition algorithm that runs in $O(2^k \cdot |V|^2) + O(|V|) = O(2^k \cdot |V|^2)$ time, where k is the most number of edges need to be deleted to make the graph bipartite. However, there remains one more issue. Generally, the input graphs are large and may not be able to be made bipartite with a small value of k. Fortunately, network processing programs are relative small. Moreover, the BAGs are typically sparse so there is good chance that we can apply the algorithm

```
input: G = {V, E}
output: Part[|V|], i.e., vertex set of bipartite graph
variable: Part[|V|]: int eger
        Gain[|V|][2]: real //Gain[i][j] = g(v_i, V_j}
        State[|V|]: bool // false = unused, true = used
        History[|V|][2]: int eger
        Temp[|V|]: int eger
Algorithm:
1.Part[i] = 0
2.for i = 0 to |V| - 1 do
   State[i] = 0;
   for j = 0 to 1 do
      Gain[i][j] = g(v_i, V_j)
   od
od
3.for i = 0 to |V| - 1 do
   3.1 select unused v_d such that g(v_d, V_l) = max_{j,p}(Gain[j][p]); // l ∈ {0,1} v_d ∈ V_m
   3.2 State[d] = 1;
   3.3 Part[d] = l;
   3.4 History[i][0] = d;
       History[i][1] = m;
   3.5 Temp[i] = Gain[d][l];
   3.6 for j = 0 to |V| - 1 do
         if State[j] = 1
            continue;
         if Part[j] = m
            Gain[j][l] = Gain[j][l] - 2W(v_d, v_j);
         else if Part[j] = l
            Gain[j][m] = Gain[j][m] + 2W(v_d, v_j);
      od
4. select t to maximize G = Σ_{j=1}^{t} Temp[j];
5.if (G > 0)
      for j = t + 1 to |V|
         Part[History[j][0]] = History[j][1];
      goto 2;
```

Fig. 4. Heuristic balanced bipartite graph algorithm.

on them after pruning the input graph according to the properties of bipartite graph. The basic idea is that we delete an edge e if deleting the edge e separates the graph into 2 disconnected components. For example, deleting the edge e in Fig. 5 will divide the graph into 2 disconnected components G1 and G2. These edges are called *edge separators*. Such edges certainly may not induce an odd cycle. The optimal solution of the original graph will be the union of those of G1 and G2.

The graph separation is very useful to get balanced bipartite graph because smaller graphs will have

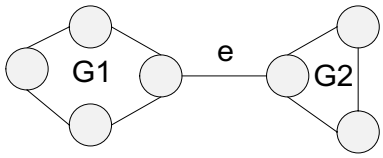

Fig. 5. Example of a graph separator.

a smaller size of cut set so that our FPT algorithm will possibly may used. At the same time, the runtime spent on separated graphs will be significantly reduced. Supposed the graph is equally divided, the runtime will decrease by 50% of the original time, i.e., from $O(2^k \cdot |E|^2)$to $O(2 \cdot 2^k \cdot (|E|/2)^2) = O(2^{k-1} \cdot |E|^2)$. Moreover, the subgraphs may contain less edges than the k we predefine, which further reduces the runtime.

To find such separators, we choose an edge, delete it from the graph and then use a depth-first-search (DFS) algorithm to test if there is a path from its one end to another. If there is no such a path, the edge is a separator. It takes $O(|E^2|)$ time to test all the edges. We color the graph with 2 colors during the DFS process. If we find a bipartite connected subgraph, we delete it from G since it is conflict-free. This can decrease the size of the graph. If the graph is still too big that there may be little chance we can make it bipartite by a small number of k edges, we will perform more complex pruning proposed in [7]. At the worst, if a cut set of size smaller than k cannot be found, we use the heuristic [1] to provide an approximate solution at polynomial time instead.

5. Balanced bipartite graph based register allocation for IXP

The algorithms presented in Section 4 can generate balanced bipartite graphs for the register allocation. The FPT algorithm is able to work out an optimal solution but spends more time. Therefore, the FPT can work well only for programs with small sizes. For large programs, instead, we may use the heuristic algorithm. In this section, we investigate how to assign registers for IXPs.

Register allocation for IXPs needs to consider not only assigning registers but also assigning banks. In summary, there are three kinds of methods: first registers next banks (FRNB), first banks next registers (FBNR) and mixed registers and banks (MRB). FBNR is easy to implement and can resolve bank conflicts effectively. However, FBNR can not judge the register pressure in advance so that it has a difficulty in balancing the register pressure, which often causes a lot of spilling codes [33]. On the other hand, MRB cannot effectively coordinate the synchronization between the register allocation and the bank allocation. The worst situation is all adjacent vertexes of a vertex are conservatively assigned in a bank, which also cause a lot of unnecessary spilling codes.

We use the first method, i.e., FRNB, because it can reduce spilling codes as much as possible. Possibly, FRNB allocation will result in more bank conflicts than FBNR. But FRNB outperforms to FBNR because code spilling needs to access memory and the cost for resoling bank conflicts is much less than that for code spilling. The framework of register allocation is shown in Fig. 6. Each phase will be presented in the following subsections, where we assume the total register number be N and each bank have N/2 registers.

5.1. Register allocation

We firstly allocate registers since it reduce the total number of code spilling, which is more expensive than the bank conflict. Interference graph is built then virtual registers are mapped into physical register

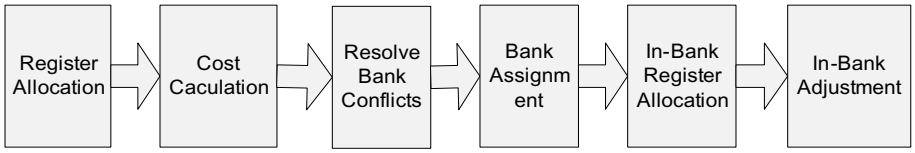

Fig. 6. Register allocation framework for IXP.

$$1.\ B1 = A1$$
$$\mathbf{A2 = A1 + A2} \Rightarrow 2.\ \mathbf{A2} = B1 + \mathbf{A2}$$

Fig. 7. Insert a move instruction.

$$\begin{array}{ll} & 1.\ A1 = A1 \oplus B1 \\ & 2.\ B1 = A1 \oplus B1 \\ & //now\ B1 = A1_{org},\ A1 = A1_{org} \oplus B1_{org} \\ \mathbf{A2 = A1 + A2} & 3.\ \mathbf{A2} = B1 + \mathbf{A2} \\ & 4.\ B1 = A1 \oplus B1 \\ & 5.\ A1 = A1 \oplus B1 \\ & //\ now\ A1 = A1_{org},\ B1 = B1_{org} \end{array}$$

Fig. 8. In-place bank exchange.

assuming a single monolithic register bank with traditional register allocation algorithm [9]. The resulting interference graph is N-colorable and each vertex presents a physical live range (PLR). Here physical live range is the maximal du-ud chain with respect to the physical register names. They can contain different virtual live ranges that reside in the same physical register.

5.2. Cost model

Before present the cost model, we first introduce two methods for resolving bank conflict.

1) If the register pressure of the program point at which the PLRs conflict is less than N/2, we use a move instruction to resolve the conflict. For example, in the left instruction in Fig. 7, A1 conflicts with A2. A move instruction, B1 = A1, is inserted, and B1, which is a register on the opposite bank of A, replaces A1 in the original instruction. Such a B1 can certainly be found since the register pressure is less than N/2, which means there is at less one register available in either bank.
2) When the number of registers is more than N/2-1, one of the bank may not have unused registers. If both the banks have at least one available register, we can resolve conflicts by a move instruction as in case (a). If not, it is possible that the opposite bank, against which the PLRs reside in, has no available register. In this case, we use a tricky technique called in-place bank exchange to avoid spills, instead of inserting a move instruction. For example, in the left instruction in Fig.8, A1 conflicts with A2. Two XOR instructions are inserted before the original one, exchanging the values of A1 and B1. Then B1 replaces A1 in the original instruction. The last two instructions restore the values of A1 and B1.
3) As to BAG, these are equivalent to delete an edge between two vertexes, since they no longer conflict. Theoretically, a new node should be added into the BAG. Nevertheless, it is not necessary since the new node only conflict with one node and such a node will never induce an odd cycle.

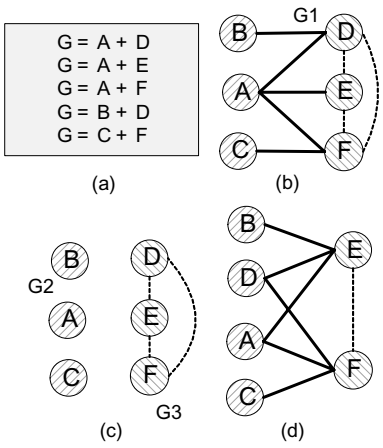

Fig. 9. Flip vertex between banks.

To estimate the cost of resolving PLRs conflicts, we define the following quantities:

1. *conf*$_i$ is the i^{th} conflict of two PLRs
2. *confwt*$_i$ is the cost of resolving *conf*$_i$. In the case (a) mentioned above, it is taken to be the cost of the move instruction, say 1. In the case (b), we don't simply set the cost to be the 4 XOR instructions, say 4, since PLRs conflict with less register pressure have better changes to be resolved as in case (a). Instead, the cost is taken to be $2^{(2p/N-2)}$, where p is the register pressure of the conflict point. Its range is [1,4] when *p is* within [N/2, N].

The cost to resolve all the conflicts within two PLRs is

$$\sum_i confwt_i \cdot 8^{depth(conf_i)}$$

5.3. Conflict resolving and bank balance

In this step, we build the weighted BAG, whose edges have weights representing the cost to remove it from the graph, i.e. the cost to resolve the conflict between two PLRs. To make a graph bipartite, we can delete vertexes and edges or split nodes. However, we cannot delete nodes here since PLRs cannot be deleted. We employ the uniform 2-way graph partition algorithm presented in Section 4 to make the graph bipartite and balance with the minimal cost. If k passes the threshold, we invoke a heuristic algorithm.

5.4. Bank assignment

At this time, bank conflict graph has been converted into balanced bipartite graph. A connected graph will keep connected after it is reduced into a balanced bipartite graph. As a result, we can use just one way to color the BAG with 2 colors. The vertexes get the same color will be assigned to the same bank.

5.5. Register assignment in a bank

Now the conflicts are resolved and the banks are assigned. The next step is to assign registers in-bank. We perform it with traditional register allocator. Though the graph is balanced, there still may be a few

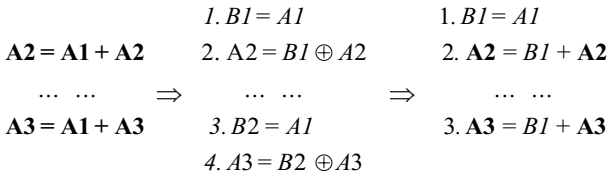

Fig. 10. Eliminate a move instruction.

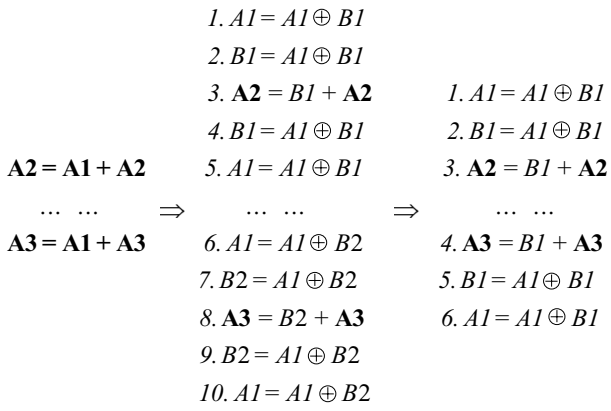

Fig. 11. Eliminate 4 XOR instructions.

cases that the registers cannot be successfully allocated in-bank. For example, for the instructions in Fig. 9(a), assume vertexes A, B and C get bank A while vertexes D, E and F get bank B in Fig. 9(b). Each bank has two registers. The original graph is divided into two subgraphs induced by the vertexes in the same bank respectively. For example, G1 is divided into G2 and G3 as shown in Fig. 9(c). We try to color each subgraph with two colors but fail to do so in bank B. Then some PLRs have to be "spilled" to the opposite bank, which will induce new conflicts. The vertex with least spill cost is to be spilled with the highest priority. The vertex to be spilled is the one with least flip cost as calculated in Section 4.2 and the new conflicts are resolved as described in Subsection 5.2. Vertex B is spilled since it has the least spill cost = 1. The final interference graph is shown in Fig. 9(d), assuming D be "spilled".

5.6. Redundant instruction elimination

A insert instruction at every conflict point may results in redundancy. For example, assume there are unused register in the opposite bank of bank A, and A1 and B1 is not redefined between the two instructions in the left of Fig. 10. In this case, two move instructions may be inserted as shown in the middle of Fig. 11. We can remove the third instruction and rename B2 to B1 in the fourth instruction.

If the register pressure is high, there is more redundancy. Also assume A1 and B1 are not redefined between the original two instructions. Four instructions can be eliminated and the result is shown in the right of Fig. 11. Simple dataflow algorithm can perform the eliminations easily.

6. Experiments and evaluation

We developed a simulation system oriented to mobile and wireless applications using Intel IXP simulation workbench. Each mobile node with an IXP processor connects with others using wireless

Table 1
System configuration parameters

Parameters	Value
K	32
The number of total threads	4
The number of registers in a bank	32
The number of used threads	1

Table 2
Benchmark properties

Applications	Number of instructions
IPSec_forward	6194
10GB_Ethernet	1781
L2_forward	2405
Firewall	566
MPLS	1025

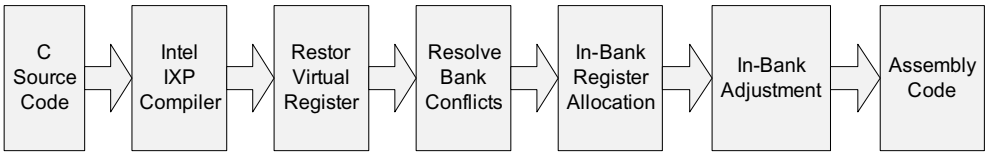

Fig. 12. Compilation flowchart.

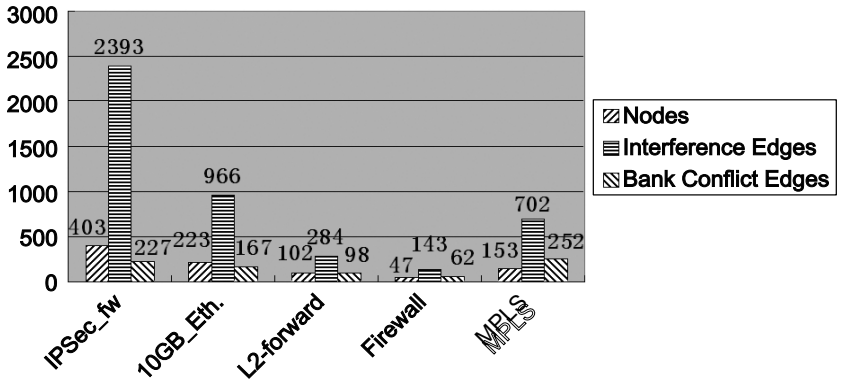

Fig. 13. Number of conflict edges and bank conflict edges.

mesh communication. The node was configured according to Table 1. The benchmarks were compiled using the C compiler for Intel IXP to generate assembly codes, restored in all the virtual registers, and assigned registers with our algorithm, as shown in Fig. 12. Then conflict graph and BAG was constructed from the assembly codes. The construction of BAG is easy since IXP only has 80 instructions. Table 2 shows the properties of the benchmark applications used in our experiments.

We used five real network applications, whose parameters are listed in Table 2 and features are shown in Fig. 13. The average vertex degree is 9.6 in the interference graph and 1.2 in bank conflict graph, respectively.

We tested our algorithms FPT and Heuristic algorithm using the above five benchmarks in terms of

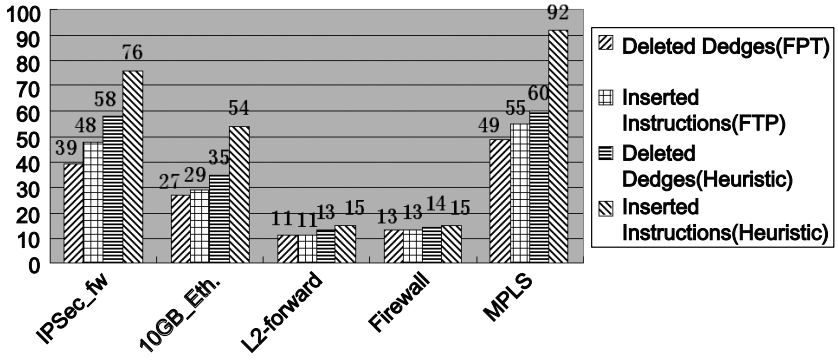

Fig. 14. Results without redundancy elimination.

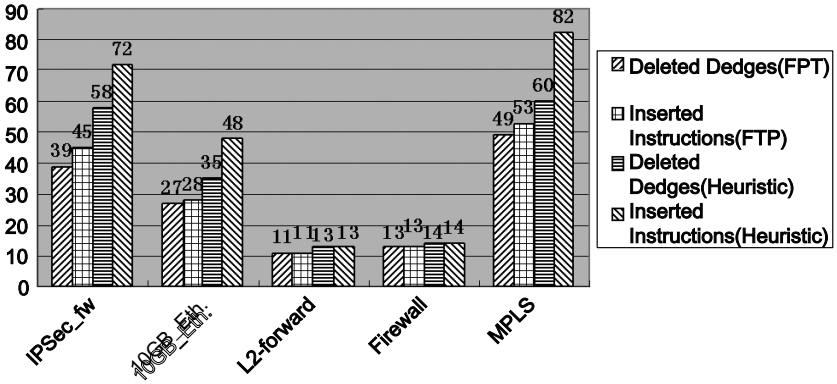

Fig. 15. Results with redundancy elimination.

the following performance metrics.

The number of deleted edges and inserted instructions. In the experiments, we tested the number of edges and instructions that need to be deleted and inserted respectively for IXP register allocation. In Fig. 14, redundant instructions in tested programs were not deleted, while Fig. 15 shows the experimental results after the redundant instructions were removed from the programs. The difference between Figs 14 and 15 is a little because live range in network applications is small, and there are only 2 or 3 conflicts. By comparison, the algorithm FPT outperformed the Heuristic one in terms of whether the number of deleted edges or inserted instructions. In FPT, the number of deleted edges always approximately equals to the number of inserted instructions, which means that FPT can solve conflicts by inserting less instructions. At the same time, Heuristic algorithm needs approximate number of deleted edges and inserted instructions for small graph such as Firewall. For Heuristic algorithm, however, the inserted instructions grow more quickly than the deleted edges with the increase of the size of programs.

Throughput. Typically, a network application consists of an unconditional loop, which continuously handles and forwards incoming packets, as shown in Fig. 16. We used OC48 (2.5 Gb/s) engine to test the throughput of benchmarks using different register allocation algorithms. The results are shown in Fig. 17 and Fig. 18. We found that the two algorithms all provide around 3.5 Gb/s network bandwidth. The processing speed may improve by 2.7% after deleting redundant codes, and the FPT is 2.5% better than the Heuristic algorithm.

```
for ( ; ; )
{
    Rx_Packet(); //receive data packets;
    ...
    IPv4_Fwd(); //forward data packets;
    ...
    Tx_Packet(); //send data packet;
}
```

Fig. 16. A typical network processing application.

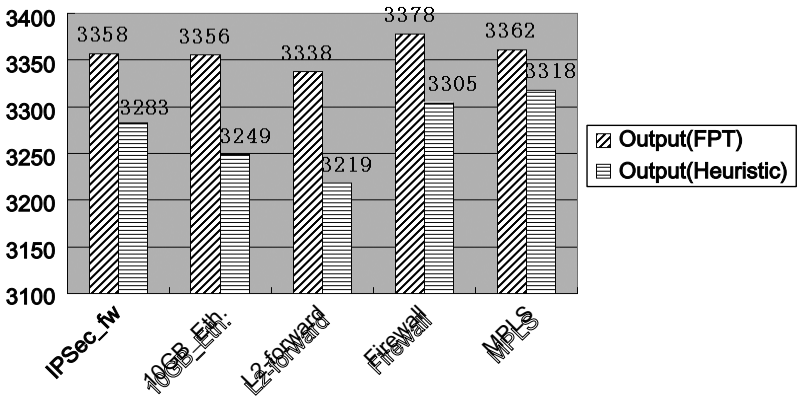

Fig. 17. Throughput without redundancy elimination.

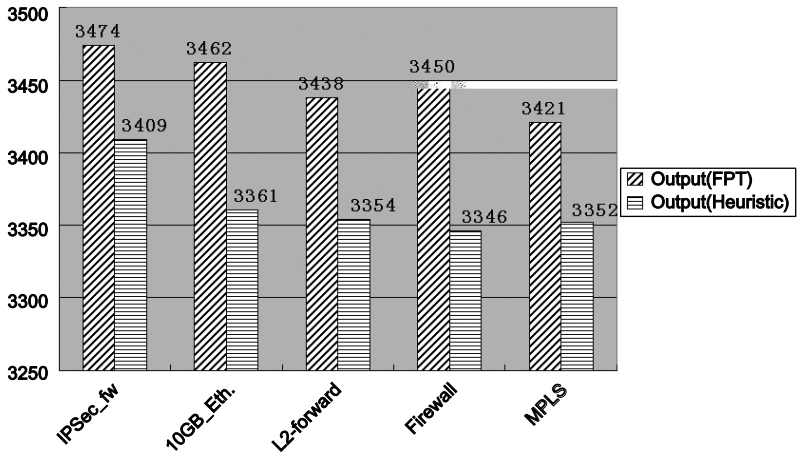

Fig. 18. Throughput with redundancy elimination.

Comparison with Post-RA. We tested the inserted instructions for our FPT algorithm, Heuristic algorithm and Zhuang's Post-RA [9], which firstly assigns registers and then banks. We tested them using the ratio of inserted instructions to the number of conflict edges in order to avoid the affects of different

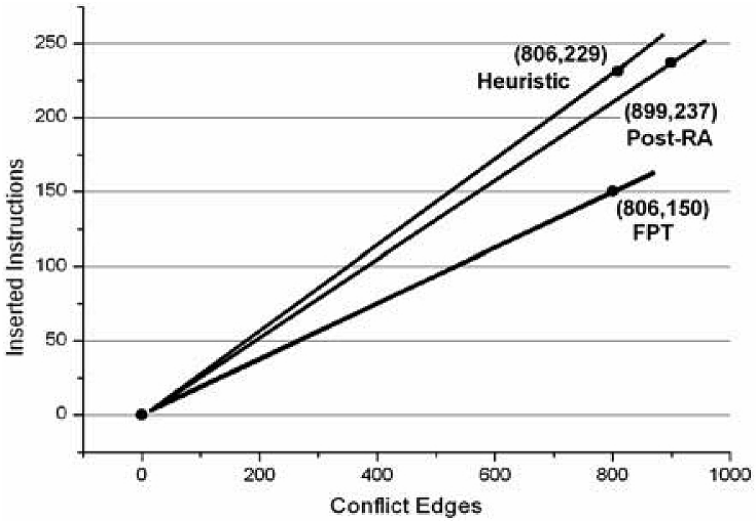

Fig. 19. Comparison among FPT, heuristic and post-RA.

benchmarks. The results are shown in Fig. 19, where Y-axis represents the sum of inserted instructions of all the benchmarks and X-axis represents the sum of total conflict edges. Our FPT algorithm needs fewer insert instructions to resolve more conflicts since FPT takes the interaction of edges as a whole and provide optimal solution instead of approximate solution. Besides, the bank pressure is almost balanced since we take a uniform graph partition algorithm. Note that the data in Fig. 19 is the sum of inserted instructions and conflict edges for all five benchmarks.

From the Fig. 19, we can draw a conclusion that the FPT is the best one in all three algorithms in terms of the ratio of inserted instructions to conflict edges. Post-RA is better than Heuristic, but worse than our FPT. Heuristic is the worst one in this experiment.

7. Conclusion

Mobile and wireless networks need to provide sufficient bandwidth to improve users' transparent experience and meet increasingly QoS demands of the users. Network processors, e.g., IXP, oriented to mobile and wireless networks typically have many registers, where effective register allocation is very important for the efficient computing ability and thus high network throughput. For this purpose, we have presented a graph uniform 2-way partition algorithm (FPT), which can work out an optimal solution to the graph partition, and a heuristic algorithm, which can get approximate results, for the generation of balanced bipartite graph. Based on the algorithms, we designed a framework for two banks based IXP register allocation. Experimental results demonstrate the framework and the algorithms are efficient in register allocation for IXP network processors. Further, our FPT algorithm is practical and effective in solving certain NP-complete problems in embedded systems. By employing such an algorithm to give an optimal solution, we can consider the BAG as a whole and reduce the inserted instructions for conflict resolving.

Acknowledgement

Feilong Tang would like to thank The Japan Society for the Promotion of Science (JSPS) and The Un-

iversity of Aizu (UoA), Japan for providing the excellent research environment during his JSPS Post-doctoral Fellow Program (ID No. P 09059) in UoA, Japan.

This work was supported by the National High Technology Research and Development Program (863 Program) of China (Nos. 2006AA01Z172 and 2008AA01Z106), and the National Natural Science Foundation of China (Nos. 60773089, 60533040 and 60725208).

References

[1] A. Agarwal, M. Charikar, K. Makarychev and Y. Makarychev, $O(\sqrt{\log n})$ approximation algorithms for min UnCut, min 2CNF deletion, and directed cut problems, *Proceedings of the 37th STOC*. ACM Press, 2005, pp. 573–581.
[2] A. Durresi and M. Denko, Advances in mobile communications and computing, *Mobile Information Systems* **5**(2) (2009), 101–103.
[3] A.W. Appel and L. George, Optimal spilling for CISC machines with few registers, *Proceedings of the ACM SIGPLAN 2001 Conference on Programming Language Design and Implementation*, 2001, pp. 243–253.
[4] B.C. Cheng, H. Chen and R.Y. Tseng, Context-Aware Gateway for Ubiquitous SIP-Based Services in Smart Homes, *Proceedings of International Conference on Hybrid Information Technology* (*ICHIT '06*), 2006, pp. 374–381.
[5] C.H. Lee, M. Kim and C.I. Park, An efficient k-way graph partitioning algorithm for task allocation, in parallel computing systems, Proceedings of the First International Conference on Systems Integration, 1990, pp. 748–751.
[6] D.J. Kolson, A. Nicolau, N. Dutt et al., A method for register allocation to loops in multiple register filearchitectures, *Proceedings of The 10th International Parallel Processing Symposium* (*IPPS '96*), pp. 28–33.
[7] F. Hüffner, N. Betzler and R. Niedermeier, Optimal edge deletions for signed graph balancing, *Proceedings of the 6th Workshop on Experimental Algorithms* (*WEA'07*), 2007, pp. 297–310.
[8] F. Zhou, J.C. Zhang, C.Y. Wu and Z.Q. Zhang, A Register Allocation Framework for Banked Register Files with Access Constraints, *Proceedings of The 10th Asia-Pacific Computer Systems Architecture Conference*, 2005, pp. 269–280.
[9] G.J. Chaitin, Register allocation and spilling via graph coloring, *Proceedings of the SIGPLAN Symposium on Compiler Construction* **39**(4) (1982), 66–74.
[10] G. Mino, L. Barolli, F. Xhafa, A. Durresi and A. Koyama, Implementation and performance evaluation of two fuzzy-based handover systems for wireless cellular networks, *Mobile Information Systems* **5**(4) (2009), 339–361.
[11] G.Y. Lueh, T. Gross and A. Adl-Tabatabai, Fusion-based register allocation, *ACM Transactions on Programming Languages and Systems*, 2000, pp. 431–470.
[12] IXP 2800 Network Processor: Programmer's Reference Manual, Dec. 2001. http://www.intel.org.
[13] J. Abella and A. Gonzãlez, On Reducing Register Pressure and Energy in Multiple-Banked Register Files, *Proceedings of IEEE International Conference on Computer Design* (*ICCD'03*), 2003, pp. 14–20.
[14] J. Ahn, Novel directory service and message delivery mechanism enabling scalable mobile agent communication, *Mobile Information Systems* **4**(4) (2008), 333–349.
[15] J. Guo, J. Gramm, F. Hüffner, R. Niedermeier and S. Wernicke, Improved fixed-parameter algorithms for two feedback set problems, *Proceedings of 9th Workshop on Algorithms and Data Structures* (*WADS'05*), 2005, pp. 158–168.
[16] J. Fu, O. Hagsand and G. Karlsson, Queuing Behavior and Packet Delays in Network Processor Systems, *Proceedings of the 15th International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems*, 2007, pp. 217–224.
[17] L. George and M. Blume, Taming the IXP network processor, *ACM SIGPLAN Conference on Programming Language Design and Implementation*, 2003, pp. 26–37.
[18] J. Hiser, S. Carr, P. Sweany and S.J. Beaty, Register assignment for software pipelining with partitioned register banks, *Proceedings of 14th International Parallel and Distributed Processing Symposium* (*IPDPS 2000*), pp. 211–217.
[19] J. Park, J. Lee and S. Moon, Register Allocation for Banked Register File, *Proceedings of the 2001 ACM SIGPLAN workshop on Optimization of middleware and distributed systems*, pp. 39–47.
[20] J. Wagner and R. Leupers, C compiler design for a network processor, *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* **20**(11) (2001), 1302–1308.
[21] M. Adiletta, M. Rosenbluth, D. Bernstein et al., The Next Generation of Intel IXP Network Processors, *Intel Technology Journal* **6**(3) (2003), 6–18.
[22] M.K. Chen, X.F. Li, R. Lian et al., Shangrila: Achieving high performance from compiled network applications while enabling ease of programming, *ACM SIGPLAN Conference on Programming Language Design and Implementation*, 2005, pp. 224–236.
[23] M. Safar, H. Sawwan, M. Taha and T. Al-Fadhli, Virtual social networks online and mobile systems, *Mobile Information Systems* **5**(3) (2009), 233–253.

[24] P. Briggs, K. Cooper and L. Torczon, Rematerialization, *Proceedings of ACM SIGPLAN Conference on Programming Language Design and Implementation*, 1992, pp. 311–321.
[25] P. Brink, M. Castelino, D. Meng, C. Rawal and H. Tadepalli, Network processing performance metrics for IA- and IXP-Based Systems, *Intel Technology Journal* **7**(4) (2003), 77–91.
[26] R. Collins, F. Alegre, X.T. Zhuang and S. Pande, Compiler assisted dynamic management of registers for network processors, *Proceedings of 20th International Parallel and Distributed Processing Symposium* (*IPDPS 2006*), 2006, 10 pages.
[27] R.G. Downey and M.R. Fellows, Parameterized Complexity, *Springer*, 1999.
[28] P. Montesinos, W. Liu and J. Torrellas, Using register lifetime predictions to protect register files against soft errors, *Proceedings of the 37th Annual IEEE/IFIP International Conference on Dependable Systems and Networks* (*DSN '07*), 2007, pp. 286–296.
[29] P. Raghavan, A. Lambrechts, M. Jayapala et al., Very Wide Register: An Asymmetric Register File Organization for Low Power Embedded Processors, *Proceedings of Design, Automation & Test in Europe Conference & Exhibition* (*DATE '07*), 2007, pp. 1–6.
[30] S. Izumi, K. Yamanaka, Y. Tokairin et al., Ubiquitous supervisory system based on social contexts using ontology, *Mobile Information Systems* **5**(2) (2009), 141–163.
[31] S. Jang, S. Carr, P. Sweany and D. Kuras, A Code Generation Framework for VLIW Architectures with Partitioned Register Files, *Proceedings of International Conference on Massively Parallel Computing Systems*, 1998.
[32] T. Monreal, A. Gonzalez, M. Valero et al. Delaying physical register allocation through virtual-physical registers, *Proceedings of the 32nd Annual International Symposium on Microarchitecture* (1999), 186–192.
[33] X. Zhuang and S. Pande, Resolving Register Bank Conflicts for a Network Processor, *Proceedings of the 12th International Conference on Parallel Architectures and Compilation Techniques* (*PACT 2003*), page. 269.
[34] X. Zhuang and S. Pande, Balancing register allocation across threads for a multithreaded network processor, *Proceedings of ACM SIGPLAN 2004 Conference on Programming Language Design and Implementation* (*PLDI 2004*), 2004, pp. 289–300.
[35] Z. Liu, H. Che, K. Zheng et al. A trace driven study of the effectiveness of cache mechanism for network processors, *Proceedings of International Conference on Wireless Communications, Networking and Mobile Computing* **2** (2005), 978–981.

Feilong Tang received his Ph.D degree in Computer Science and Technology from Shanghai Jiao Tong University (SJTU), China in 2005. He has been working with the Department of Computer Science and Engineering at SJTU, China since 2005. From September 2007 to October 2008, Dr.Tang was a visiting researcher in the University of Aizu, Japan. Currently, Dr. Tang is a Postdoctoral Research Fellow of Japan Society for the Promotion of Science (JSPS). His research interests include embedded systems, parallel and distributed computing, wireless sensor networks, grid and pervasive computing and reliability computing.

Ilsun You received his MS and Ph.D. degrees in the Division of Information and Computer Science from the Dankook University, Seoul, Korea in 1997 and 2002, respectively. He is now an assistant professor in the School of Information Science at the Korean Bible University. His research interests include MIPv6 security, key management, authentication and access control. He is a member of the IEEK, KIPS, KSII, and IEICE.

Minyi Guo received his Ph.D. degree in computer science from University of Tsukuba, Japan. He is now a full professor at the Department of Computer Software, The University of Aizu, Japan. His research interests include pervasive computing, parallel and distributed processing and parallelizing compilers. He is a member of ACM, IEEE, IEEE Computer Society, and IEICE.

Song Guo received the PhD degree in computer science from the University of Ottawa, Canada in 2005. From 2006 to 2007, he was an Assistant Professor at the University of Northern British Columbia, Canada. He is currently an Assistant Professor at School of Computer Science and Engineering, the University of Aizu, Japan. His research interests are in the areas of protocol design and performance analysis for communication networks, with a special emphasis on wireless ad hoc and sensor networks.

Long Zheng received the B.S Degree in computer science and technology from Huazhong University of Science and Technology (HUST), China, in 2006 and M.S. Degree in computer science and engineering from the University of Aizu, Japan, in 2009. He is now a Ph.D. student at School of Computer Science and Engineering, the University of Aizu, Japan. He is also in Dual Degree Program of HUST from 2007. His research interests include chip multiprocessor, parallel and distributed processing, pervasive computing, and P2P.