Register Allocation for Hybrid Register Architecture in Nonvolatile Processors

Yiqun Wang * Hongyang Jia * Yongpan Liu * Qing'an Li † Chun Jason Xue † Huazhong Yang *

{ypliu,yanghz}@tsinghua.edu.cn
* Nanoscale Integrated Circuit & System Laboratoray
Dept. of EE Tsinghua Univ. Beijing

jasonxue@cityu.edu.hk
† Department of Computer Science
City University of Hong Kong, Hong Kong

***Abstract*—Nonvolatile processors (NVP) have been an emerging topic in recent years due to its zero standby power, data retention and instant-on features. The conventional full replacement architecture in NVP has drawbacks of large area overhead and high backup energy. This paper provides a partial replacement based hybrid register architecture to significantly abate above problems. However, the hybrid register architecture can induce potential critical data loss and backup errors. In this paper, we propose a critical-data overflow aware register allocation (CORA). Different from other register allocation methods, CORA efficiently reduces the possibility of critical data spilling and backup errors. The experiment results show that CORA reduces the critical data overflow rate by up to 52%. The hybrid register architecture reduces the chip area by 45.1% and backup energy by 82.8% when using CORA.**

I. Introduction

Nonvolatile processor (NVP) is one of the most promising techniques to realize energy efficient computing systems. Some previous works use different nonvolatile memories to implement NVP [1]–[3] and they declares the following advantages in NVP: (i) zero-standby power, (ii) nanosecond sleep/wakeup speed [2], (iii) high resilience to power failures, (iv) micro-second-grained power management supported [3].

Previous works always use a full replacement architecture in NVP, which means that all the registers are replaced by the nonvolatile flip-flops (NVFF). However, people tend to use large register files in ultra low power processors [4] and GPUs [5] (e.g. the area of registers can exceed 40% of the total chip area [4]). In this case, the full replacement architecture induces nontrivial problems. First, it increases the chip area by up to 90% [6], due to the large area of NVFF. Second, it consumes relatively high backup energy, which increases the risk of backup failure when power supply is weak.

This paper proposes a hybrid register architecture (shown in Fig. 1(c)) to tackle the problems of the full replacement architecture. It partially replaces the registers to reduce the number of NVFFs, hence the area and backup energy. The rationality of this architecture is that the critical data, which represent the system states, are always stored in a small portion of flip-flops. However, this architecture has the risk of unexpected critical data loss, i.e., some critical data are not assigned to the NVFFs when the backup proceeds. In order to prevent this situation, we introduce a novel register allocation algorithm to minimize the opportunity of critical data overflow. It is called critical-data overflow aware register allocation (CORA). CORA aims to minimize the possibility of critical data overflow in the hybrid register architecture, compared with other register allocation algorithms which aim to minimize the memory access [7], [8].

The contributions of this paper are listed as the following:

This work was supported in part by the NSFC under grant 61204032 and 61271269, and High-Tech Research and Development (863) Program under contract 2013AA013201.

1) We propose a hybrid register architecture in NVP, which greatly reduces chip area and backup energy.
2) We develop a new register allocation algorithm named CORA, which achieves rather low critical data overflow rate and backup error rate compared with other register allocation algorithms.
3) The experiment results show that CORA reduces the critical data overflow rate by up to 52%, and reduces the chip area by 45.1% and backup energy by 82.8% in the hybrid register architecture.

The remaining of the paper is organized as follows: Section II describes the pros and cons of hybrid register architecture. Section III provides the formulation of register allocation problem. Section IV describes the CORA in detail. Section V shows the comparison results between CORA and other register allocation algorithms, and Section VI concludes the paper.

II. Hybrid Register Architecture

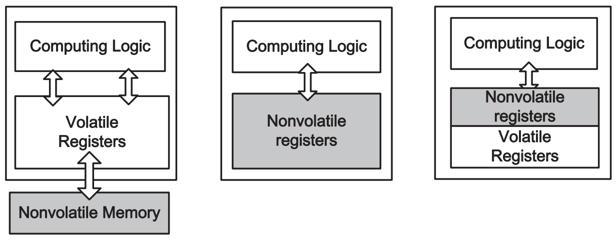
This section firstly compares three different backup architectures on area, backup speed and energy. The advantages of proposed hybrid register architecture are shown. Then, some special problems of the hybrid register architecture are introduced.

A. Backup architecture comparison

The backup architecture helps to backup the system states from volatile registers to nonvolatile memories. In some register-dense system [4], [5], the data migration is always costly, so the optimization of backup architecture is very important.

Three different backup architectures are shown in Fig. 1. The centralized nonvolatile memory architecture [9] (see in Fig. 1(a)) is not a good candidate, because the global data movement between registers and standalone nonvolatile memory suffers from limited bandwidth and high backup overhead. The full replacement NVP (see in Fig. 1(b)) has a large number of NVFFs which induces tremendous area overhead. For example, in an 8051 microcontroller, the full replacement architecture requires 1607-bit NVFFs and increases the chip area by 90% [6]. Comparatively, the hybrid register architecture (see in Fig. 1(c)) only replaces a part of volatile registers with nonvolatile ones. It has a comparable fast backup speed with the full replacement architecture, while reducing chip area and backup energy significantly.

We made some quantification to estimate the three backup architectures in a general purpose 8051 NVP core. We assume that the total register file is 128-Byte and the number of NVFF is 256. Experiment shows that the hybrid register architecture can increase backup speed and reduce backup energy by 2-3 orders of magnitude, and only increases the chip area by 8%, compared with centralized nonvolatile memory architecture. Compared with full replacement architecture, the hybrid register architecture can reduce the chip area by 43% and



(a) Centralized nonvolatile memory architecture (b) Full replacement architecture (c) Hybrid register architecture

Fig. 1. The comparison of three different memory architectures for the register data backup

backup energy by 75%, and achieve comparable backup speed. In sum, from the vision of area, speed and energy, the hybrid register architecture is more efficient.

B. Special problems of hybrid register architecture

Above results are based on an ideal assumption that all the critical data can be contained in the 256-bit NVFFs. It is not always reachable in real cases. When the size of critical data exceed the number of NVFFs, some critical variables must be spilled to volatile registers or memories. This situation is defined as ”Critical Data Overflow”.

When the critical data overflow and and backup operation happen simultaneously, some critical data are lost and the backup can cause system error. We call this situation the ”Backup error”. In order to prevent the system error, the processor cannot process the current backup operation, and roll back to its previously backup states when power is on. To achieve this mechanism, the compiler must set an additional indicator to alarm the critical data overflow event. However, the backup errors and the following rollbacks can induce nontrivial performance loss when the critical data overflow rate is high. In order to lower the critical data overflow rate, the compiler must allocate the critical variables to nonvolatile registers as much as possible. Since the traditional compilers treat all the data uniformly, a new register allocation method is required distinguish the critical data. In the following sections, we will discuss the design of the register allocation aiming to minimize the critical data overflow rate.

III. Register allocation Problem

In this section, we give the overview of the register allocation problem. The formulation of register allocation problem and the discussion of critical parameters are illustrated. Then we point out the design challenges.

A. Critical data overflow aware register allocation

The formulation to minimize the critical data overflow time is described as the following:

Given Sets and Variables:

- $\mathbf{V}$: the set of all variables,
- $\mathbf{V}_c$: the set of noncritical variables,
- $\mathbf{I} = \{I_1, I_2, I_3, ..., I_{max}\}$: The set of instruction index of a program.
- $L(V_k)$: the life interval of variable V_k expressed in instruction index range $[i_p, i_q]$. $(i_p, i_q \in \mathbf{I}, V_k \in \mathbf{V})$

Variables for optimization:

- $Al(V_k, i)$: the allocation result function, which returns the register index where variable V_k allocates when executing instruction $i \in \mathbf{I}$.

Objective:

- Minimize the total time of critical variable overflows: T_{ov}

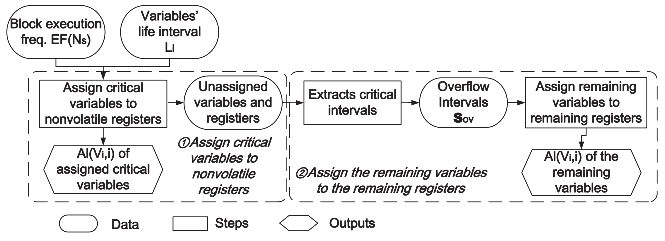

Block execution freq. EF(Ns)
Variables' life interval Li
Assign critical variables to nonvolatile registers
Unassigned variables and registers
Extracts critical intervals
Overflow Intervals Sov
Assign remaining variables to remaining registers
Al(Vi,i) of assigned critical variables
①Assign critical variables to nonvolatile registers
②Assign the remaining variables to the remaining registers
Al(Vi,i) of the remaining variables
Data
Steps
Outputs

Fig. 2. The proposed CORA register allocation flow

Intuitively, T_{ov} is the union of life intervals of overflowed critical variables. To give this formulation, we consider the life interval of critical variable $L(V_k)$ separately. Firstly, we calculate the overflowed interval $Ov(L(V_k))$ during each life interval of critical variables:

$$Ov(L(V_k)) = \{i \in \mathbf{I} | i \in L(V_k), Ov(i, V_k) = 1\}, V_k \in \mathbf{V}_c \quad (1)$$

where $Ov(i, V_k) = 1$ means V_k overflows at instruction i.

Therefore, the overall overflow range is the union set of each $Ov(L(V_k))$:

$$R_{ov} = \bigcup (Ov(L(V_k)), V_k \in \mathbf{V}_c, \quad (2)$$

and the total overflow time is

$$T_{ov} = Intv2Time(R_{ov}). \quad (3)$$

The function $Intv2Time()$ translates the instruction index interval to the real time interval based on the execution time and frequency of instruction i. From Equations (2) and (3), we can see that the total overflow time T_{ov} is considered as the sum of all overflowed intervals. Therefore, we should firstly reduce the number of overflows, then reduce the time of each overflows, to achieve small T_{ov}. In section IV, we consider those two reductions in the proposed algorithm.

B. Design challenges of register allocation algorithm

The design challenges of register allocation algorithm are embodied as following aspects: i, we introduce the function $intv2Time()$ to translate the instruction interval to time interval, so we need to consider the translation in our algorithm; ii, the function $intv2Time()$ is not in an analytical form, so we need a heuristic solution to solve T_{ov}; iii, the hybrid register architecture consists of different memory structures with different access speed, and the variables are classified to two groups, thus the algorithm should allocate variables to volatile and nonvolatile memories accordingly.

IV. Algorithm Description

This section describes the proposed register allocation algorithm – CORA. We firstly provide the overall algorithm flow of CORA. Then we specify its two main sub-flows.

A. Allocation algorithm flow

The proposed register allocation flow is shown in Fig. 2. The input $EF(N_s)$ is the execution frequency of instruction block N_s in the control flow graph (CFG). $EF(N_s)$ is used to calculate the function $Intv2Time()$ in Equation (3). The value of $EF(N_s)$ can be obtained by existing methods [10]. The following flow includes two main sub-flows. The first sub-flow only allocates the critical variables to NVFFs in order to minimize the critical data overflow. The second sub-flow allocates the remaining variables to registers and helps to reduce the penalty of each overflow from the first sub-flow. These two sub-flows jointly reduce the total overflow overhead. They are specifically described as the following.

B. Assign critical variables to nonvolatile registers

It is similar to the "Greedy Allocator" [11] in LLVM [12] when we assign critical variables to nonvolatile registers. The principle is to assign the life interval to the physical register with the greatest allocation priority [11]. The allocation priority is defined as the variable's usage density, in order to prevent spilling the most active variables to the memory. By contrast, we proposed a new priority metric to reduce the critical variable overflow.

Algorithm 1: Assigning critical variables to nonvolatile registers

```
Input: $\{L_i\}$:The set of original life interval of critical variables;
       $\{F_j\}$:The set of original registers' free segments;
       $\{EF(N_s)\}$: The set of execution frequency of instruction
       block $N_s$ in CFG
Output: $Al(V_n, i)$ of assigned critical variables;
Variables: $\mathbf{S}_{life}$,$\mathbf{S}_{free}$
1 $\mathbf{S}_{life} = \{L_i\}$;
2 $\mathbf{S}_{free} = \{F_j\}$;
3 $\forall i, j\ p_{i,j} = f(\{EF(N_s)\}, L_i, F_j), L_i \in \mathbf{S}_{life}, F_j \in \mathbf{S}_{free}$;
      /* Calculate priority matrix $\mathbf{P} = (p_{i,j})$ */
4 while $\exists p_{i,j} > 0$ do
5     Find the largest priority $p_{i0,j0}$;
6     Assign and Split the life interval $L_{i0}$ to the free segment $F_{j0}$;
7     Update register allocation results $Al(V_n, i)$;
8     Update $\mathbf{S}_{life}$,$\mathbf{S}_{free}$;
9     Update priority matrix $\mathbf{P}$;
```

The algorithm of this sub-flow is shown in Algo.1. We firstly build up two sets, $\mathbf{S}_{life}$ and $\mathbf{S}_{free}$, to store the unassigned life intervals (L_i) of critical variables, and free segments (F_j) of nonvolatile registers (Line 1-2). Then we calculate the priority of each pair (L_i, F_j) using the function $f(\{EF(N_s)\}, L_i, F_j)$ and build up the priority matrix **P** (Line 3). Afterwards, if positive priority values exist (Line 4), we choose the largest value $p_{i0,j0}$ (Line 5), and assign corresponding life interval L_{i0} to the free segment F_{j0} (Line 6). We consider the splitting of L_{i0} in the assignment. Then, we update the critical variables $Al(V_n, i)$ (Line 7), and the $\mathbf{S}_{life}$, $\mathbf{S}_{free}$ and matrix **P** are updated (Line 8-9). We repeat the priority based assignment (Line 5-9) until none of elements in **P** is positive which means no better allocation (L_i, F_j) exists.

In Algo.1, the most important step is defining the priority $p_{i,j}$. In order to formulate $p_{i,j}$, we firstly define some special parameters. We denote x_k as index of instruction block where instruction k locates, and the boolean function $\delta_{i,k}$ indicates whether the instruction k is in the live interval L_i. Intuitively, if the overlap time between L_i and F_j is longer, $p_{i,j}$ is larger. The overlap time can be expressed as

$$t_{i,j} = \sum_{k=start(F_j)}^{end(F_j)} EF(N_{x_k})\delta_{i,k}, \tag{4}$$

where

$$\delta_{i,k} = \begin{cases} 0, & k \in L_i \\ 1, & k \notin L_i, \end{cases} \tag{5}$$

and the $start(F_j)$ and $end(F_j)$ indicates the index of the first instruction and the last instruction in free segment F_j.

Meanwhile, $p_{i,j}$ should have negative relationship splitting cost denoted as C_{split}. Therefore, the $p_{i,j}$ can be expressed as

$$p_{i,j} = t_{i,j} + C_{split} = \sum_{k=start(F_j)}^{end(F_j)} EF(N_{x_k})\delta_{i,k} + C_{split}, \tag{6}$$

where the C_{split} has negative value. The splitting cost is generated at the splitting point and the transition point between instruction blocks covered by split life interval.

As discussed above, this sub-flow can greedily choose the (L_i, F_j) with the longest overlap time. From this vision, this algorithm can reduce the opportunities of critical variable overflow to a great extent. However, this algorithm cannot handle the duration of each overflow. We will deal with this problem in the next sub-flow.

C. Assign the remaining variables

This sub-flow executes a similar priority based allocation algorithm to assign the remaining life intervals. Different from the previous flow, the pairs (L_i, F_j) include all the unassigned variables and all the remaining registers. The target is to minimize the time of the overflow segments. Therefore, the priority $p_{i,j}$ is defined according to this target.

Since the memory access contributes most time cost, the $p_{i,j}$ should consider the "spilling to memory" during the overflow segments. However spilling-to-memory overhead should be predicted before register allocation is done. To achieve the prediction, we introduce β_k and γ_k. β_k indicates whether the instruction k is in overflow intervals and γ_k indicates the potential spiling-to-memory time of instruction k. Then $p_{i,j}$ can be expressed as

$$p_{i,j} = \sum_{k=start(F_j)}^{end(F_j)} EF(N_{x_k})\delta_{i,k}\beta_k\gamma_k + C_{split}, \tag{7}$$

where the $start(F_j)$, $end(F_j)$, $\delta_{i,k}$ and C_{split} have the same definition as in Equations (4) and (6).

Corresponding to the Equations (2) and (3), this sub-flow can reduce the $Intv2Time(L(V_k))$ of each $L(V_k)$. Therefore, after the two sub-flows, the T_{ov} can be significantly reduced, and the results will be shown in the Section V.

V. Experimental Results

The performance of CORA on hybrid register architecture is evaluated in this section.

A. Experiment setup

In the experiment, we use LLVM [12] to obtain the block frequency and life intervals. The results of the actual critical variable overflow are obtained by an in-house program simulator. The testing programs include three common programs from Mibench: **dijkstra**, **qsort** and **CRC32**, and three programs with typical patterns: sequential program, branch program and loop program. The critical data in each program is randomly selected. In real cases, the critical data depend on the application requirements.

Two other register allocation algorithms are used for comparison:

1) Linear Scan (LS): the prototype of basic linear scan algorithm [13].
2) Refined Linear Scan (RLS): the refined linear scan algorithm based on LS, adding the distinction of critical and noncritical variables.

The number of registers is set to 16 in the experiments. We do not use large number of registers because the testing programs do not contain a large number of interleaved variables. However, the results of limited registers can be easily extended to large number of registers.

B. Critical data overflow rate

The critical data overflow rate is defined as the critical overflow time overs the total execution time. The statistics and comparison results are shown in Table I, where 25% registers are nonvolatile.

The last two columns reveals that CORA can reduce up to 74.8% overflow rate versus LS, and 51.9% versus RLS in the best cases. CORA can still provide 23.1% and 16.1% overflow rate reductions

TABLE I
EVALUATION OF CRITICAL DATA OVERFLOW RATE

R.A. Algorithms		CORA	LS	RLS	Overflow Reduction	
					vs. LS	vs. RLS
Ov. rates	**CRC32**	0.252	0.709	0.525	64.4%	51.9%
	qsort	0.657	0.854	0.841	23.1%	21.8%
	dijkstra	0.121	0.478	0.164	74.8%	26.4%
	Sequential	0.666	0.943	0.794	29.4%	16.1%
	Loop	0.546	0.984	0.872	44.5%	37.4%
	Branch	0.592	0.963	0.871	38.5%	32.0%

in the worst cases. For each row, we can see that RLS is always better than LS, because it distinguishes the critical/noncritical data and nonvolatile/volatile registers. However, our proposed CORA still has better results than RLS, because it targets at overflow minimization.

C. Backup error estimation

This section shows the backup error estimation under different patterns of power failures. We assume two patterns from two real scenarios of energy harvesting. One is solar power failure which meets the poisson process with the expectation λ. The other is the vibration power supply failure which occurs periodically with the period P_{pf}. The results of backup error rates are shown in Table II.

TABLE II
EVALUATION OF BACKUP ERROR ESTIMATION UNDER DIFFERENT POWER FAILURE PATTERNS

Algorithm		Backup error rate under solar power			Backup error rate under vibration power		
		$\lambda = 0.005$	$\lambda = 0.025$	$\lambda = 0.05$	$P_{pf} = 10$	$P_{pf} = 50$	$P_{pf} = 100$
CORA		0.012	0.064	0.125	0.252	0.046	0.018
Linear Scan		0.039	0.181	0.353	0.709	0.156	0.065
Refined Linear Scan		0.027	0.127	0.260	0.525	0.078	0.071
Backup error Reduction	vs. LS	69.2%	64.6%	64.6%	64.5%	70.5%	72.3%
	vs. RLS	55.6%	49.6%	51.9%	52.0%	41.0%	74.6%

1 The testing program is under program "**CRC32**".
2 The λ is the sunlight blockage probability. The unit of P_{pf} is instruction cycles.

Table II shows the backup error rate increases when λ increases or P_{pf} decreases because of more frequent power failures. From the last two columns, we can see CORA can provide more than 64% backup error reduction compared with LS and more than 41% compared with RLS. The effect of reducing backup failure is even better than reducing the critical data overflow rate, because CORA can both reduce the overflow flow rate and shorten each overflow's endurance.

D. Area and Backup energy

Chip area and backup energy eventually conclude the effect of the register allocation algorithm and hybrid register architecture. This experiment compares them under three backup architectures (see in Fig. 1). We also list the results under different allowed overflow rates. To meet the requirement of large number of registers, we use the input program as a combination of all testing programs. The results are shown in Table III

TABLE III
EVALUATION OF CHIP AREA AND BACKUP ENERGY REDUCTION COMPARED WITH FULL REPLACEMENT

Algorithms	Area Reduction		Backup Energy Reduction	
	$R_{ov} = 0.2$	$R_{ov} = 0.4$	$R_{ov} = 0.2$	$R_{ov} = 0.4$
CORA	44.1%	45.1%	80.9%	82.8%
LS	36.1%	40.1%	67.0%	73.5%
RLS	42.1%	42.6%	77.2%	78.1%

1 The testing program is a combination of all testing programs and the variables in total is 284.
2 R_{ov} stands for the overflow rate.

From Table III, the hybrid register architecture can reduce chip area by 36.1% and reduce backup energy by 67.0% compared with full replacement. According to the corresponding results of $R_{ov} = 0.2$ and $R_{ov} = 0.4$, we can see that the looser demand of overflow rate results in better area and energy performances, because it requires less nonvolatile registers. Moreover, the proposed CORA can provide additional 3% area reduction and 4% backup energy reduction compared with RLS (even more with LS), which means the proper register allocation algorithm can further improve the performance of hybrid register architecture.

VI. CONCLUSIONS

This paper presents a hybrid register architecture to reduce the area and backup energy overhead in NVP with large register files. A corresponding register allocation algorithm named CORA is proposed to tackle the critical data overflow problem of the hybrid register architecture. Experiment results show that CORA can reduce the critical data overflow rate by 52%, and reduces the chip area by 45.1% and backup energy by 82.8% along with the hybrid register architecture. Our future work will aim to improve the compiling speed of CORA.

REFERENCES

[1] W. Yu, S. Rajwade, S. Wang, B. Lian, G. Suh, and E. Kan, "a non-volatile microcontroller with integrated floating-gate transistors," in *Proceedings of the 5th Workshop on Dependable and Secure Nanocomputing*. ACM Press, 2011, pp. 1–4.
[2] S. C. Bartling, S. Khanna, M. P. Clinton, S. R. Summerfelt, J. A. Rodriguez, and H. P. McAdams, "An 8mhz 75ua/mhz zero-leakage non-volatile logic-based cortex-m0 mcu soc exhibiting 100¡400ns wakeup and sleep transitions," in *ISSCC 2013*, 2013, pp. 432–433.
[3] Y. Wang, Y. Liu, S. Li, D. Zhang, B. Zhao, B. Sai, M.-F. Chiang, Y. xin Yan, and H. Yang, "A 3us wake-up time nonvolatile processor based on ferroelectric flip-flops," in *ESSCIRC 2012*, september 2012, pp. 149–152.
[4] P. Meinerzhagen, S. Sherazi, A. Burg, and J. Rodrigues, "Benchmarking of standard-cell based memories in the sub-vt domain in 65-nm cmos technology," *Emerging and Selected Topics in Circuits and Systems, IEEE Journal on*, vol. 1, no. 2, pp. 173–182, 2011.
[5] S.-L. Chu, C.-C. Hsiao, and C.-C. Hsieh, "An energy-efficient unified register file for mobile gpus," in *Embedded and Ubiquitous Computing (EUC), 2011 IFIP 9th International Conference on*, 2011, pp. 166–173.
[6] Y. Wang, Y. Liu, Y. Liu, D. Zhang, S. Li, B. Sai, M.-F. Chiang, and H. Yang, "A compression-based area-efficient recovery architecture for nonvolatile processors," in *DATE 2012*, march 2012, pp. 1519 –1524.
[7] Y. Huang, T. Liu, and C. Xue, "Register allocation for write activity minimization on non-volatile main memory," in *Design Automation Conference (ASP-DAC), 2011 16th Asia and South Pacific*, 2011, pp. 129–134.
[8] T. Liu, A. Orailoglu, C. Xue, and M. Li, "Register allocation for simultaneous reduction of energy and peak temperature on registers," in *Design, Automation Test in Europe Conference Exhibition (DATE), 2011*, 2011, pp. 1–6.
[9] M. Zwerg, A. Baumann, R. Kuhn, M. Arnold, R. Nerlich, M. Herzog, R. Ledwa, C. Sichert, V. Rzehak, P. Thanigai, and B. Eversmann, "An 82 ua/mhz microcontroller with embedded feram for energy-harvesting applications," in *ISSCC 2011*. IEEE, feb. 2011, pp. 334 –336.
[10] Y. Wu and J. Larus, "Static branch frequency and program profile analysis," in *MICRO 1994*, 1994, pp. 1–11.
[11] Jakob Stoklund Olesen, "LLVM Project Blog: Greedy Register Allocation in LLVM 3.0," *Website: http://blog.llvm.org/2011/09/ greedy-register-allocation-in-llvm-30.html.*
[12] C. Lattner and V. Adve, "Llvm: A compilation framework for lifelong program analysis & transformation," in *CGO 2004*. IEEE, 2004, pp. 75–86.
[13] M. Poletto and V. Sarkar, "Linear scan register allocation," *ACM Transactions on Programming Languages and Systems (TOPLAS)*, vol. 21, no. 5, pp. 895–913, 1999.