Novel Online Data Allocation for Hybrid Memories on Tele-health Systems

Longbin Chen, Meikang Qiu[1], Wenyun Dai

Pace University, New York, USA
Email: {longbin.chen, mqiu, wenyun.dai}@pace.edu

Houcine Hassan

Universitat Politcnica de Valncia, Valencia, Spain
Email: husein@disca.upv.es

Abstract

The developments of wearable devices such as *Body Sensor Networks* (BSNs) have greatly improved the capability of tele-health industry. Large amount of data will be collected from every local BSN in real-time. These data is processed by embedded systems including smart phones and tablets. After that, the data will be transferred to distributed storage systems for further processing. Traditional on-chip SRAMs cause critical power leakage issues and occupy relatively large chip areas. Therefore, hybrid memories, which combine volatile memories with non-volatile memories, are widely adopted in reducing the latency and energy cost on multi-core systems. However, most of the current works are about static data allocation for hybrid memories. Those mechanisms cannot achieve better data placement in real-time. Hence, we propose online data allocation for hybrid memories on embedded tele-health systems. In We present dynamic programming and heuristic approaches. Considering the difference between profiled data access and actual data access, the proposed algorithms use a feedback mechanism to improve the accuracy of data allocation during runtime. Experimental results demonstrate that, compared to greedy approaches, the proposed algorithms achieve 20%-40% performance improvement based on different benchmarks.

Keywords:
Hybrid memory, non-volatile memory, tele-health, data allocation, dynamic programming, heuristic approach

1. Introduction

Tele-health is a novel solution for efficient and long-distance health care services. According to a report by InMedica [1], the global market of tele-health will hit 6-billion by the year 2020. Patients can start video chats with doctors across the country and receive diagnoses remotely. Improvements in *wireless sensor networks* (WSNs) add more features to tele-health services. Patients wear intellectual physiological sensors, which detect health conditions such as *electrocardiography* (ECG), blood pressure, and glucose. A *body sensor network* (BSN) is a cluster of these sensors. It collects signals of sensors and forwards them to doctors with telecommunications. In tele-health, patients can make more flexible medical schedules while doctors can obtain more accurate health care information from patients.

Embedded devices are widely used in tele-health systems due to their unmatched convenience and increasingly computing power. Signals collected from a local BSN can be transmitted to a mobile device of the same patient. The mobile device processes the signals and forwards it to remote medical organizations. Patients can also schedule medical meetings and set up remote diagnoses by using their mobile devices. All these activities are supported by wireless technologies such as WiFi and GPRS.

On embedded systems, compared to single-core solutions, multi-core systems with parallelism can be more energy-efficient for complex tele-health applications. For example, two cores which have the same total size and energy consumption as a large single core can improve the performance of the single core by 40% [2]. However, energy cost is still a critical issue for mobile devices with limited battery life, so an energy-aware design of memory hierarchies becomes important to mobile tele-health systems.

[1]Dr. Meikang Qiu is the corresponding author

Embedded systems, especially those with SRAM-based cache memories, are suffered from both power leakage and space issues [3][4][5][6]. The caches occupy about 50% of the total chip areas [7]. SRAM, as a widely used on-chip memory, has several drawbacks: first, it consumes large on-chip area, due to its low density. SRAM uses six-transistor architecture. Second, SRAMs have high power leakage issues. Magnetic RAM (MRAM) [8] has been gathering wide interests for various appealing characteristics, such as high density, fast access speed, and excellent non-volatility [9][10][11][12]. Unlike traditional RAMs which store information by charging, MRAM uses Magnetic Tunnel Junctions (MTJs). A MTJ consists of two magnetic metal layers and stores information based on the relative orientation of two layers magnetization. However, MRAM also faces the challenge of limited writes and relatively high write latency

Zero-capacitor RAM (Z-RAM) [13][14] is another attractive memory technology that can overcome the high cost of SRAM and MRAM with relatively few performance degradation. Z-RAM is equipped with only one transistor while SRAM uses six transistors. Therefore, it can achieve much higher density compared to SRAM. The biggest disadvantage of Z-RAM is their non-volatility as SRAM and DRAM, and their relatively long read/write latency. ZRAM is manufactured with only one transistor instead of six transistors used in SRAM. Therefore, they can afford much higher density (usually 5x) than SRAM.

This work is based on a previous conference version [3]. The major contributions of this version include:

1. We propose a hybrid on-chip memory architecture with SRAM, MRAM, and ZRAM. This is a novel combination of memories on tele-health systems.
2. We propose two heuristic algorithms for data allocation: simulated annealing and hill climbing.
3. We adopt a feedback mechanism to reduce the impact of inaccurate data profiling.
4. We set up simulated experiments to verify proposed algorithms.

The rest of this paper is organized as: Section 2 describes rationale of our approaches and proposed hybrid memory architecture. Section 3 gives concrete examples of the proposed algorithms. Section 4 and 5 present the online dynamic algorithm ODPDA. Section 6 presents the experimental results and Section 7 introduces research works related to hybrid memory systems. Section 8 concludes the paper.

2. System architecture

In mobile tele-health systems, data placement means how tele-health data is stored in on-chip memories or the main memory of mobile devices. This is based on the access number of each data item and the capacities of on-chip memories and the main memory. Generally, there are two types of data placement: *global* placement and *regional* placement. The *global* placement means, during the execution of a tele-health application, there is only one fixed data placement for the whole application. In *regional* placement, the application is divided into regions which can be executed in parallel. Before the execution of each region, compilers insert instructions to store data in on-chip memories. *Global* placement solutions have the benefit of simpleness, but they fail to take advantage of data locality for each region. So they cannot achieve minimized total cost.

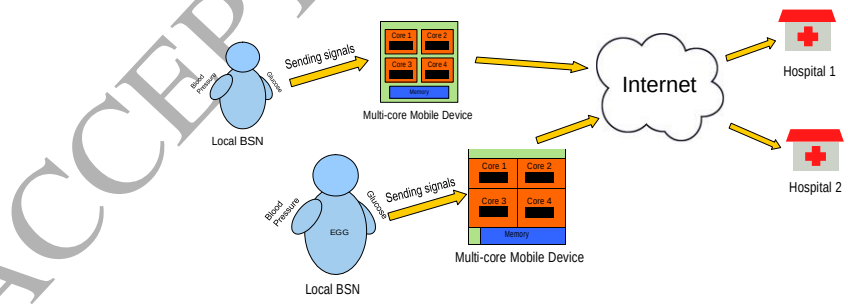


Figure 1: System Overview. Local *body sensor networks* (BSNs) and hospitals are connected via Internet. BSNs send signals to hospitals, while hospitals provide services. SPM stands for scratch-pad memory, which is a type of on-chip memory.

As shown in 1, in a tele-health system, the BSNs send information of patients to embedded devices such as smart phones. The embedded devices then process and forward the information to remote hospitals. We propose our hybrid memories

for the multi-core embedded systems to reduce the latency and energy cost of memory accessing. The adoption of hybrid memories in multi-core systems has drawn attention of researchers. A hybrid main memory with DRAM and *Phase Change Memory* (PCM) was proposed in the work of J. Meza et al. [15], where the authors used a large DRAM cache for the nonvolatile main memory. Their work focused on designing a fine granularity migration to reduce memory access latency. W. Tian et al. [16] proposed task allocation on hybrid Nonvolatile memory (NVM). Authors presented both offline and online heuristic allocation algorithms. They considered the size and writes of NVMs. They achieved optimal solutions with an integer linear programming (ILP) model. Many research works have been done on optimized data allocation for hybrid scratchpad memories (SPMs). M. Qiu et al. [17] proposed a hybrid SPM simulator for evaluation. Their work focused on application-specific single-core systems.

As shown in Fig. 2, in a multi-core system, each core has hybrid memories which consist of SRAM, MRAM, and ZRAM. DRAM is used as off-chip memory. The cores are connected via an on-chip interconnection. We use local access to represent a core accesses data on its hybrid memories [18]. Remote access represents a core accesses data on other cores hybrid memories. All cores connect to the off-chip memory through a shared bus. The contentions for on-chip interconnection and off-chip memory access are beyond the scope of our work. Therefore, we assume the cores can access data remotely without being interfered by other cores. In this architecture, the time and energy costs of accessing main memory are the highest, while the cost of remote access is higher than the one of local access.

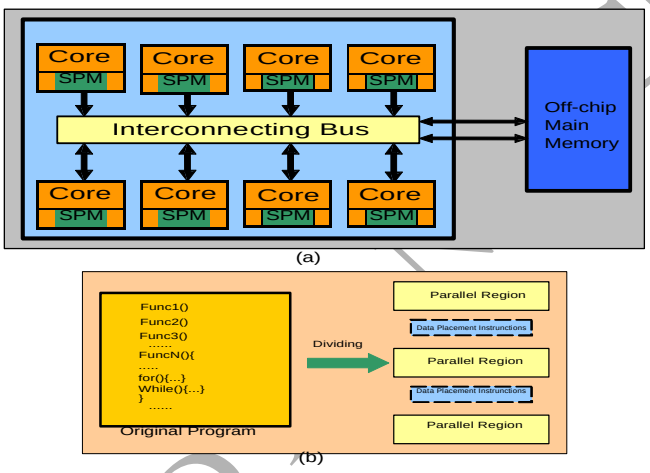


Figure 2: Hardware and software model: (a) an eight-core system with on-chip memories (b) demonstration of parallel regions.

Profiling is a process where a compiler collects information from data segments. The compiler can use the information to insert data movement instructions between program regions. In our work, we profile the access numbers of each data items in a program. The granularity of data in on-chip allocation represents the size of each data item that the compiler will adopt to profile information. A fine granularity will increase the accuracy of profiling process and consequently lead to a cost-efficient on-chip data allocation. However, Fine granularity also yields overheads of profiling and increases the workload of compilers.

However, the granularity of profiling is another critical issue. Currently, most researchers categorize the profile process into two categories: global profiling and regional of data item for the whole program. The regional profiling divides the program into small regions and obtains the data access information for each region. The global profiling costs fewer overheads, but it loses the benefit of fine granularity profiling. The regional profiling takes the advantage of data locality. Hence, the profiling result will be more accurate, and the data placement method will be more efficient. As shown in Fig. 2, when a program is divided into regions properly, the compiler will insert data placement instructions before the execution of each region.

Moreover, the profiled information of data access might not be different from the actual information during runtime. Feedback information will be a critical factor to minimize the inaccuracy of profiling. In our proposed algorithm, as shown in Fig. 2, we include feedback information in the calculation of data access cost array. The feedback factor represents the differences between profiled data access times and actual data access times for one core. We collect the feedback factor for each core. Before the execution of each region, we will insert instructions which consider the feedback factor to reallocate the position of data items.

3. Motivations

In the paper, we propose dynamic programming and heuristic approaches to minimize the number of memory access of programs running on tele-health systems. In our design, these programs are running on a hybrid memory system that consists of SRAM, MRAM, and ZRAM.

In this section, we illustrate the proposed algorithms with simple examples. Too make the examples more understandable, we include greedy algorithm. As shown in Table 1, the data capacity of SRAM, ZRAM, and MRAM are 2, 4, and 2 respectively. In the example, we assume DRAM has sufficient memory spaces for all data items. The read latency of SRAM, ZRAM, MRAM, and DRAM are 0.58ns, 0.71ns, 0.85ns, and 8.5ns respectively while the write latency of these memories are 0.58ns, 4.5ns, 1.4ns, and 12ns respectively. Since energy consumption for memory access is in proportional to latency, we only consider memory latency in this example.

	SIZE	READ(ns)	WRITE(ns)
SRAM	2	0.58	0.58
ZRAM	4	0.71	4.5
MRAM	2	0.85	1.4
DRAM	INF	8.5	12

Table 1: The access latency of each type of memory

The proposed work is based on multi-core systems and every running program is divided into several regions with feed-back factors. However, in this section, for simplicity, only one region and a single core is considered. In Table 2, 15 data items are used for data allocation. The number of reads and writes for each data item are listed in the table.

	A	B	C	D	E	F	G	H
Read	14	18	23	5	14	4	9	8
Write	18	2	1	19	9	2	24	16
	I	J	K	L	M	N	O	
Read	5	26	21	24	8	8	15	
Write	22	10	13	23	11	1	18	

Table 2: The number of reads and writes for each data item

In a greedy approach, the data item with most read/write will be placed on the on-chip SRAM. The rest data items will be placed on SRAM based on the order of number of write. Therefore, data *L* will be placed on SRAM first and data *G* will be the second one placed on SRAM. The process continues until no space left on SRAM. Then MRAM is used to place data items with most write access. After that, the rest data items will be placed on ZRAM until no space is left on ZRAM. Eventually, the data items remaining are stored on the main memory, which is the DRAM.

The eventual data allocation of this greedy approach is shown in the following table. For example, the memory access cost for data item *L* can be calculated as: data $L = 24 \times 0.58\text{ns} + 23 \times 0.58\text{ns} = 27.26\text{ns}$.

	Data Items	Total memory access cost
SRAM	L, G	1.7ms
ZRAM	D, I	3.8ms
MRAM	A, H, O, K	5.9ms
DRAM	B, C, E, F, J, M, N	24.8ms

Table 3: A greedy data allocation

In the proposed dynamic programming, for each data item, the possible data placement and the cost of the placement are

calculated and stored in a "dictionary". The each data placement and its cost can be reused directly in the calculation of the next data item. In this manner, no redundant calculation will occur. Therefore, the computing time can be largely reduced.

In this example, data item *A* is the first one to be calculated. The latencies of placing data *A* on SRAM, ZRAM, MRAM, or DRAM are calculated and stored in a multiple dimensional array. For the next data *B*, since each memory has enough space left, it can be placed on SRAM, ZRAM, MRAM, or DRAM. The latencies of data item *B* on each memory can be expressed as: $LTE_B = LTE_{A,S} + LTE_{B,S}$ or $LTE_{A,S} + LTE_{B,Z}$ or,..., or $LTE_{A,D} + LTE_{B,D}$, where $LTE_{A,S}$ and $LTE_{B,Z}$ mean the latency of placing data *A* on SRAM and placing data *B* on ZRAM. Therefore, the minimum latency of allocating data *A* and *B* is: $Min(LTE_{A,S} + LTE_{B,S}, LTE_{A,S} + LTE_{B,Z}, ..., LTE_{A,D} + LTE_{B,D})$.

	A	B	C	D	E	F	G	H
SRAM	14	18	23	5	14	4	9	8
ZRAM	18	2	1	19	9	2	24	16
MRAM	5	26	21	24	8	8	15	
DRAM	22	10	13	23	11	1	18	

Table 4: The cumulative table for dynamic programming

As shown in Table 4, for $i \in 1, 2, 3, ..., N$, the memory latency for *i* data items is stored in a latency "dictionary", which is a multiple-dimensional array in this section. Therefore, the minimum latency for *i* data items will be the minimum sum of the pre-calculated latency of *i* - 1 data items and the latency of current data item.

3.1. Heuristic Approach

In the proposed heuristic approach, initially all data items are placed by their occurring order. Data item *A* and *B* are placed on SRAM, *C*, *D*, *E* and *F* are placed on ZRAM, while *G* and *H* are placed on MRAM. The rest data items are placed on Main Memory - the DRAM. The initial time latency of this data allocation *T* can be calculated as: $T = T_{r,s} \times a_r + T_{w,s} \times a_w + ... + T_{w,mm} \times o_w = 0.58ns \times 14 + 0.58ns \times 18 + ... + 8.5ns \times 15 + 12ns \times 18$.

After the initialization, the data allocation *DA* will be stored as the best solution. Then a copy of DA will be created. Then in the copy, two data items are swapped randomly. For example, data *O* is swapped from main memory to ZRAM and data *F* is swapped from ZRAM to main memory. The time latency of the new data allocation is re-calculated. Since the new data allocation has less total latency, we save the new data allocation and its latency as the best solution.

To minimized duplicate operations, we keep a list of the recent swapped data items. For example, data *O* and *F* are kept in the list after the previous steps. In the next N steps, data *O* and *F* are excluded from swapping, where N can be parameter to tune the performance of the algorithm. In this example, we simply repeat the swapping process for a large number of times, such as 10000. In heuristic approaches, the repeating number is a key parameter for tuning performance. We can improve the efficiency of our approach by adding another condition: if a solution is the best data allocation for a consecutive number of *C*, we conclude that this solution is the best solution.

4. ODPDA Algorithm

In this section, we present our Online Dynamic Programming for Data Allocation (ODPDA) algorithm in detail. We will first build an allocation cost table, which presents the time latency of each data item when it is placed on different memory locations. Then, with the help of this table, we describe the procedures of ODPDA algorithm. After that, we will explain the core idea of this algorithm in detail.

4.1. Data Access Cost

The ODPDA algorithm tracks the access cost of each data item. In this paper, we use fixed values to present local cost and remote cost for data items. The fixed values are obtained from HP Labs [19]. If the the data item is located in main memory, then each core has the same high access cost for the data. If the data item is stored in a core, then the access cost consists of local cost and remote cost. In hybrid memories, even inside the same core, the access cost might be different. It depends on the different type of memory where the data item is stored. We use a two-dimensional array named CST to represent the cost distribution among memories. The rows of the array consist of data items, while the columns mean the memories on each

core. For example, CST (i,j) represents the total access cost of data *i* when it is allocated on core *j*. CST (i,j) includes the local access cost of core *j* and the remote access cost of other cores. With this cost array, we can directly obtain the total cost of an individual data placement.

4.2. Dynamic Programming

Even though we build the cost array that reflects the cost distribution of all possible data allocation solutions. It is still a complex problem to achieve the minimized total access cost of all data items, since the number of placement combination grows in a exponential way as the numbers of data item and core increase. To reduce the cost of our mechanism, we apply a dynamic programming in our solution. Dynamic programming is widely used to solve complex problems. The key concept of it is to break a problem into smaller subproblems, and use the results from subproblems to obtain the final solution.

In the proposed method, we assume a function *dataAlloc* represents the final result of the data allocation, which is the allocation with minimized total data access cost. Function can be expressed as: dataAlloc(D_i, $C_{1,S}$,$C_{1,M}$,...,$C_{n,Z}$), where D_i represents a set of *i* number data items. $C_{1,S}$, $C_{1,M}$, and $C_{n,Z}$ mean the available space slots on core 1's SRAM, core 2's MRAM, and core n's ZRAM, respectively. In dynamic programming, the function *dataAlloc* can be expressed as: dataAlloc(D_i) = dataAlloc(D_{i-1}) + CST_i, where dataAlloc(D_{i-1}) is the smaller subproblem and CST_i is the cost of ith data item.

In this equation, dataAlloc(D_i, $C_{1,S}$...) means the optimized data allocation for the first i 1 data items when data i will be placed on core 1s SRAM, and $CST_{i,1}$ represents the cost of allocating data i on core 1. Our algorithm will calculate all possible solutions recursively and obtain the minimum solutions. One critical feature of the dynamic programming is that we can reuse the results from subproblems.

4.3. Feedback Factor

Algorithm 4.1 Feedback function

Require: Program region N, data access cost function CST(i, m), profiled data access number PRF(d, c), actual data access ACT(d, c)
Ensure: : Feedback factor for each core FBF(i, c)

```
while N ≥ 0 do
    for C_j ∈ cores do
        FBF(i, c) = PRF(i, c) / ACT(i, c)
    end for
end while
return Feedback factor FBF()
```

As we mentioned, all data placement instructions are based on the profiled information such as data access times. However, during the execution, there might be differences between the profiled numbers and the actual numbers. To improve the accuracy of our algorithms in real-time, we propose a feedback mechanism in our method. The feedback factor $FBF_{i,j}$ = $PrfC_{i,j}/ActC_{i,j}$, where $FBF_{i,j}$, $PrfC_{i,j}$, and $ActC_{i,j}$ represent the factor for data i on core j, the profiled number, and the actual number respectively. For example, for data 1 on core 1, if the profiled access number is 20 and the actual access number is 32, then the feedback factor of core 1 is 0.625. In the next region, when the compiler estimates the data access number of core 1, it will include the factor.

To continue reducing the inaccuracy of data placement, we use a weighted feedback factor mechanism. We assign different weights to finished regions and achieve the final feedback factor for the next region. Assuming there are factors of core 1 from the finished four regions: FBF_1, FBF_2, FBF_3, FBF_4, we assign the weights of 1, 1, 2, and 4 to them respectively. Hence, when calculating the 5th region for core 1, we use a factor: $FBF_5 = 1/8 \times FBF_1 + 1/8 \times FBF_2 + 1/4 \times FBF_3 + 1/2 \times FBF_4$. The key principle in our mechanism is that we depend more on the most recently finished feedback factor.

4.4. ODPDA

We propose a two-phase algorithm. In the first phase, we create the feedback factors for each region. In the second phase, we use the dynamic programming model to obtain the optimized data allocation for each region. We consider the feedback factors in the calculation of data access cost array.

Algorithm 4.2 Online Dynamic Programming for Data Allocation

Require: Program region N, data access cost function CST(i, m), Feedback factor for each core FBF(i, c), profiled data access PRF(D, C), minimum time cost function Min()

Ensure: : Optimized data allocation for each region

```
1: while N ≥ 0 do
2:   for D_i ∈ PRF(D,C) do
3:     for C_j ∈ PRF(D,C) do
4:       CST(D, M) = FBF(D, C) × CST(D, M)
5:     end for
6:   end for
7:   dataAlloc(D, C...)
8:   if D_i = 1st data item then
9:     T_M M ← CST(D_i, MM)
10:    T_S 1 ← CST(D_i, C_1,s)
11:    T_Z 1 ← CST(D_i, C_1,z)
12:    T_M 1 ← CST(D_i, C_1,m)
13:    ...
14:    T_Zn ← CST(D_i, C_n,z)
15:    return Min(T_M M, T_S 1, T_Z 1,..., T_Z n)
16:  else
17:    T_M M ← dataAlloc(D_i−1, C_1,s,...,C_n,z) + CST(i, MM)
18:    if C_1,s - 1 ≥ 0 then
19:      T_S 1 ← dataAlloc(D_i−1, C_1,s - 1,...,C_n,z) + CST(i, C_1,s)
20:    end if
21:    if C_1,m - 1 ≥ 0 then
22:      T_M 1 ← dataAlloc(D_i−1,C_1,s,C_1,m - 1,...,C_n,z) + CST(i, C_1,m)
23:    end if
24:    if C_1,z - 1 ≥ 0 then
25:      T_Z 1 ← dataAlloc(D_i−1,C_1,s,C_1,z - 1,...,C_n,z) + CST(i, C_1,z)
26:    end if
27:    ...
28:    if C_n,z - 1 ≥ 0 then
29:      T_Z n ← dataAlloc(D_i−1,C_1,s, C_1,z,...,C_n,z - 1) + CST(i, C_n,z)
30:    end if
31:    return Min(T_M M, T_S 1, T_Z 1,..., T_Z n)
32:  end if
33: end while
```

According to the built recursive formulations, we describe scribe the *Online Dynamic Programming Data Allocation*-(ODPDA) algorithm in Algorithm 4.2. The input of the ODPDA algorithm is N data blocks obtained by profiling tools, the constructed allocation cost table, and the total cost table. The output of the algorithm is the minimum total cost (latency or energy consumption) for the N data blocks.

In line 1, a while loop is used for each region of the running program. In Line 4, we calculate the feedback factor FBF(i, C_j), where i and C_j means the data item i and core j respectively. The PRF(i, C_j) represents core js profiled access number of data i. In Line 1-6, we revise the data access cost table CST with the feedback factors from function FBF(D, C), where D, M, and C means the data item, the type of memory, and core respectively. Thereafter, in Line 7-21, we perform our main function dataAlloc() recursively. In Line 7-13, we initialize the situation when the first data item is allocated. In Line 15-20, we break the original data allocation into smaller data allocation. Time complexity: We can see that the time complexity of this algorithm is determined by the recursive part, which is $O(N \times size(C_{1,S}) \times size(C_{1,M}) \times size(C_{1,Z}) \times \cdots \times size(C_{N,Z}))$. Due to the limited on-chip memory space of the CMP system, the size of each hybrid memory is generally small. Assuming the size for each memory is M, for the system with P cores, the time complexity approximates $O(N \times K^{3M})$. Since M and P are constant for the given architecture, the algorithm can be solved in polynomial time.

5. Heuristic Data Allocation Approaches

Although dynamic programming can provide optimal data allocation on embedded systems, it also requires polynomial time complexity for calculating optimal data allocation. In practice, even polynomial time complexity can cause extra large latency to the program. Besides, the temporary memory space required by dynamic programming is also large. Therefore, in this paper, we propose two heuristic algorithms (Simulated Annealing and Hill Climbing) to allocate data items onto different memory slots.

In the Simulated Annealing, we use profiled data as the input of the function. For example, we use a constant such as 0 as the frozen temperature, the profiled number of data access as the frozen temperature, and the number of data items as the freezing speed. There is an initial data allocation for all data items. To calculate a near-optimal solution, we swap the positions of two random data items. Then the total energy and time costs of two data allocations are compared. If the swapped one yields less energy and time cost, it will be kept as the current optimal solution. We repeat this process until certain constraints are reached. The details of this algorithm is shown in Algorithm 5.1.

In line 1-3, an initial data allocation is generated by a random function. Each data item is assigned to a memory slot by the function. The memory slots can be SRAM, ZRAM, MRAM, or the main memory. In line 4-6, the time cost of the initial data allocation is calculated by a function called TIME. The initial time cost is stored in two variables for further steps. In line 7-8, the initial temperature and frozen temperature is obtained as the input of the algorithm.

The main part of the proposed simulated annealing algorithm is consisting of two loops from line 9-27. In the outer loop, the pseudo temperature is decreasing by a factor of SP until the pseudo temperature reaches the threshold of f. The inner loop is used to make sure sufficient comparison and swap operations to find the best solution in a reasonable amount of time. In line 11-12, two data items are selected by a random function RAND. Then the memory locations of the two data items are swapped. In line 14-24, the time cost of the new data allocation is re-calculated. The difference between new time cost the current time cost is stored in a variable. A probability *prb* is generated by random function RAND.

In line 17, if new time cost is less than current time cost, then the new data allocation is the better solution. Then new time cost will replace current time cost in line 18. In another case, if the new time cost is worse, a random probability *prb* is used to compared with the probability number generated by function EXP. If the probability *prb* is smaller, then worse time cost will be kept. In line 19-21, if the current time cost is less than the best time cost, the best time cost is updated and the new data allocation replaces current data allocation. In line 25-26, the initial temperature declines by a speed of *SP* and the best data allocation and time cost are returned when all loops are finished.

In line 1-3, similar to Algorithm 5.1, an initial data allocation is generated randomly. In line 4, the time cost of the initial data allocation is calculated by a function called TIME. The initial time cost is stored in two copies for further steps. In line 5, the initial data allocation is kept.

The core part of the proposed hill climbing algorithm is consisting of a single loop from line 6-18. The loop number N is a parameter can be adjusted for better performance. In line 7-11, two data items are selected randomly. If those two data items have been swapped recently, the random process will repeat until a new pair of data items satisfies the constraints. The new time latency is calculated in line 12. Two data items are recorded as recent swapped items. In line 14-17, if the new time latency is lower than the old one, it will be kept. And the new allocation is stored as the best solution. If no better solution is

Algorithm 5.1 Simulated annealing data allocation

Require: The set of data item D, data access cost function CST, profiled data access number PD, Feedback factor FDF, initial Temperature T, frozen temperature F, freezing speed SP, tuning loop N, random function RAND swap function SWAP, time calculation TIME, E power function EXP

Ensure: data allocation DA, time cost TC

```
for D_i in Set D do
    data allocation DA ← Init(D_i)
end for
curr, best ← TIME(DA);
copy ← DA; f, t ← F, T
while t ≥ f do
    for l ≤ N do
        D_i, D_j ← RAND(D)
        SWAP(copy, i, j)
        Get new latency: new ← TIME(copy)
        Get latency difference: dlt ← new - curr
        Get a probability: prb ← RAND()
        if dlt ≤ 0 or EXP(-dlt ÷ t) ≥ prb × FDF then
            curr ← new
            if curr ≤ best then
                Keep best results: best ← curr, DA ← copy
            end if
        end if
    end for
    t ← t × SP
end while
TC ← best
return DA, TC
```

Algorithm 5.2 Hill climbing

Require: The set of data item D, data access cost function CST, profiled data access number PD, Feedback factor FDF, tuning loop N, consecutive number C, random function RAND swap function SWAP, time calculation TIME
Ensure: data allocation DA, time cost TC

```
1: for D_i in Set D do
2:     Init data allocation DA ← Init(D_i)
3: end for
4: Calculate time latency: best, curr ← TIME(DA)
5: copy ← DA
6: for l ≤ N do
7:     Get two data items: D_i D_j ← RAND(D)
8:     while D_i and D_j are swapped recently do
9:         Get another pair: D_i, D_j
10:    end while
11:    SWAP(copy, i, j)
12:    new ← TIME(copy)
13:    Record data item i and j
14:    if curr ≤ best then
15:        best ← curr
16:        DA ← copy
17:    end if
18:    if No better solution for C loops then
19:        quit and return DA, TC
20:    end if
21: end for
22: return DA, TC
```

found for a consecutive time, then the algorithm will terminate the loop and return current data allocation in line 19. In line 18, *C* means the number of consecutive loops. When the loop finishes, the best data allocation is returned in line 22.

6. Simulated Experiments

In this section, we set up simulated experiments to verify the performance of our algorithms. We build customized simulator to obtain memory traces for selected benchmarks. The details about the simulated environment and benchmarks are introduced in Section 6.1. We compare the performance of proposed algorithms with a greedy data allocation. Our experiments include the following:

1. The read and write latency of each benchmark. We apply greed, ODPDA, simulated annealing, and hill climbing algorithms on each benchmark
2. The computing time of each algorithm. Time complexity is a key parameter in evaluating an algorithm. We calculate the computing time of each algorithm on our simulator.

6.1. Experiment setup

In the experiment, we analyze the performance of our algorithm on benchmarks from MiBench [20]. The experiment contains two major parts: first, we run these workloads on customized simulator and obtain the memory traces for these benchmarks; second, we use those memory traces as inputs for the proposed algorithms. As a comparison, we also run the benchmarks on a greedy algorithm. We collect memory access parameters from CACTI [19]. The parameters include memory read and write latency.

There are two sets of comparisons: 8-core and 16-core. Each core has a hybrid on-chip memory with SRAM, ZRAM, and MRAM of 4KB, 8KB, and 8KB respectively. The read and write latency of each type of memories are listed in Table 5

As shown in Table 5, SRAM has the lowest read and write latency but it also has the disadvantage of low density and leakage issues. ZRAM has higher memory density and relatively low read latency but it write two times slower than SRAM.

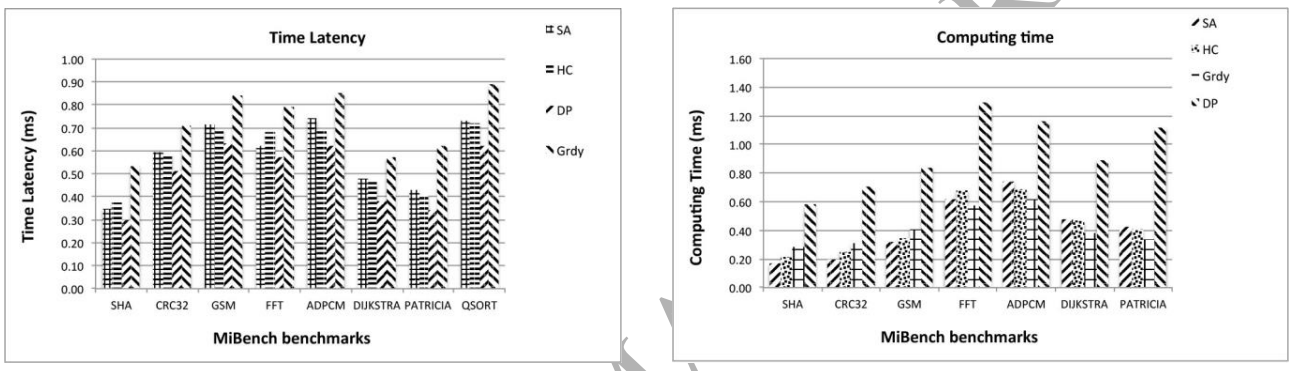
	SRAM	ZRAM	MRAM	Main
Read	0.58ns	0.85ns	0.71ns	10ns
Write	0.58ns	1.4ns	4.5ns	20ns

Table 5: Read and write latency of each memory type

MRAM can minimize energy leakage problem but it has the highest write latency except DRAM. In this experiment, we assume DRAM has sufficient memory space for all benchmarks.

We integrate these parameters into our customized simulator. To verify the effectiveness of our proposed ODPDA algorithm, we select the following benchmarks from MiBench: qsort, fft, crc32, dijkstra, adpcm, patricia, gsm, and sha.

6.2. Result Analysis



(a) (b)

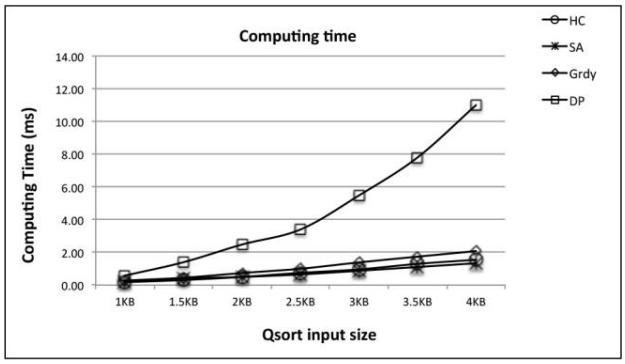
Figure 3: (a) The time latency of each algorithm on each benchmark, (b) The computing time for each algorithm on each benchmark

As shown in Figure 3a, we run each benchmark on customized simulator. The input data size of each benchmark is around 1KB. For each benchmark, we calculate the total latency of read and write operations. The number of memory access varies among selected benchmarks, hence the total latency of each benchmark is different. However, we can still observe that greedy approach results in the most memory latency in average while dynamic programming approach minimizes the total latency of benchmarks. Two heuristic algorithms, simulated annealing and hill climbing, achieve lower latency compared to greedy algorithm. The part of the simulation shows that dynamic programming can achieve the best solution while heuristic approaches can yield near-best solutions.

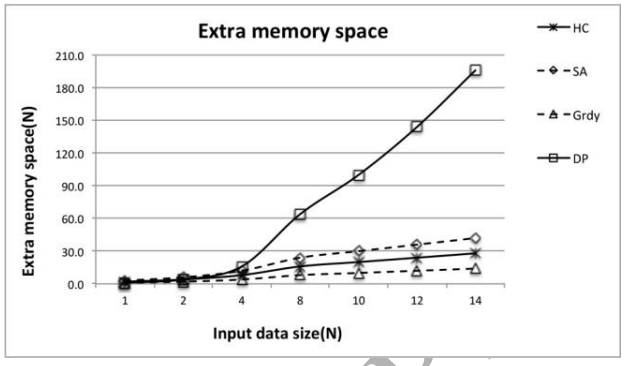
A major advantage of heuristic approaches is that they can achieve near-optimal solution in a relatively short computational time. In Figure 3b, the computing time of each approach is calculated. For a small data input (1KB), dynamic programming consumes the smallest computational time of 0.59ms for SHA benchmark, and the largest computing time of 1.29ms for FFT algorithm. As a contrast, heuristic and greedy approaches consume less computing time. Although the computing time varies among benchmarks, we can observe that dynamic programming require more computing time in average compared to heuristic and greedy approaches.

To further verify the time complexity of proposed algorithms, we select Quicksort algorithm as a single benchmark and assign it with different input data. The sizes of the input data are in the range from 1KB to 4KB. The results in Figure 4(a) shows that dynamic programming approach consumes less computing time compared to other methods.

Memory is still sparse resource on embedded systems. The proposed dynamic programming requires memory space of M $\times$ N, where M is the number of memory slots and N is the number of data items. In this experiment, we run qsort benchmark on the customized simulator with 4 cores. If each core has on-chip memory slots of M, then the total memory space needed for computation is 4MN. For heuristic approaches, simulated annealing needs three arrays with the size of N, where N is the number of data items. Hill climbing requires two of such arrays to store data allocation during computation. Hence, simulated



(a)



(b)

Figure 4: (a) The computing time for qsort benchmark, (b) The extra memory space for computing qsort benchmark

annealing requires memory spaces of 3N while hill climbing requires memory spaces of 2N only. The greedy approach only consumes one such array to sort the data items by write times.

As shown in Figure 4b, we increase the size of input data items, the space growth of greedy approach is nearly N. The space growth rates of simulated annealing and hill climbing are approximately 3N and 2N. The dynamic programming approach consumes almost 16 times of memory space. As a conclusion, the trade-off of these approaches is that dynamic programming can provide a better solution but consumes more computational time and space. Heuristic approaches can reduce computing time and space, but they might yield worse data allocation.

7. Related works

Hybrid memory systems take the advantage of volatile and non-volatile memories. Many research works have been done in hybrid memory systems. Besides proposed memories including MRAM, SRAM, and ZRAM, *phase-change memory* (PCM) is a new promising non-volatile memory technology to improve the performance and efficiency of on-chip memory systems [21, 22]. PCM has the advantage of high density and low power leakage compared to DRAM. Many researchers have applied PCM on embedded systems for better performance.

In the work of M. Qiu et al. [23], the authors applied PCM on green clouds. They presented a genetic-based optimization algorithm for chip multiprocessor systems on greed clouds. Their algorithm scheduled tasks on multi-core systems and balanced the efficiency and performance of PCM. In the work of L. E. Ramos et al [24], the authors proposed a new hybrid design that features a hardware-driven page placement policy. The policy relies on the memory controller (MC) to monitor access patterns, migrate pages between DRAM and PCM, and translate the memory addresses coming from the cores. Periodically, the operating system updates its page mappings based on the translation information used by the MC. They also proposed simulations of 27 workloads to show that their system is more robust and exhibits lower energy-delay than state-of-the-art hybrid systems. [25] presented an evaluation method to study the impacts of inaccurate execution time information to the performance of resource allocation. The authors proposed a systematic way to measure the robustness degradation of the system and evaluate how inaccurate probability parameters may affect the performance of resource allocations on CMP systems. Furthermore, they also compared the performance of three widely used greedy heuristics when using the inaccurate information with simulations.

M. K. Qureshi et al. [26] proposed an on-chip hybrid memory architecture with PCM and DRAM. In their work, PCM acts as the major storage while DRAM functions as a small buffer. The authors evaluated the benefits of such an architecture by analyze memory access latency.They conducted an experiment with a baseline system of 16-cores with 8GB DRAM. Their results showed that PCM can reduce page faults by 5X and provide a speedup of 3X in average.

Energy consumption has been one of the most critical issues in the Chip Multiprocessor (CMP). Using the Dynamic Voltage and Frequency Scaling (DVFS), a CMP system can achieve a balance between the performance and the energy-efficiency. M. Qiu et al. [27] proposed a three-phase discrete DVFS algorithm for a CMP system dedicated to applications where the period of the applications' task graph is smaller than the deadline of tasks. In these applications, multiple task

graphs are unrolled and then concatenated together to form a new task graph. The proposed DVFS algorithm is applied to the newly formed task graph to stretch tasks' execution time, lower operating frequencies of processors and achieve the system power efficiency.

Resource scheduling is one of the most important issues in mobile cloud computing due to the constraints in memory, CPU, and bandwidth. High energy consumption and low performance of memory accesses have become overwhelming obstacles for chip multiprocessor (CMP) systems used in cloud systems. In order to address the daunting memory wall problem, hybrid on-chip memory architecture has been widely investigated recently. A recent research work has been done for hybrid memories on mobile systems [28]. In their paper, the authors presented a novel hybrid on-chip memory system that consists of a static random access memory (RAM), a magnetic RAM (MRAM), and a zero-capacitor RAM for CMP systems by fully taking advantages of the benefits of each type of memory. To reduce memory access latency, energy consumption, and the number of write operations to MRAM, they also proposed a novel multidimensional dynamic programming data allocation (MDPDA) algorithm to strategically allocate data blocks to each memory.

Three-dimensional chip architecture is an interesting topic that has drawn many research attentions. CMP architectures with 3D chips provide more functionalities and higher performance. However, the high temperature on chip is a critical issue for the 3D architecture. To address this issue, J. Li et al. [29] proposed an online thermal prediction model for 3D chips. Using this model, they also proposed novel task scheduling algorithms based on rotation scheduling to reduce the peak temperature on chip. They also consider data dependencies, especially inter-iteration dependencies that are not well considered in most of the current thermal-aware task scheduling algorithms.

8. Conclusion

Tele-health technologies are promising solutions to future medical services. Hybrid memories can take the advantage of features including high density, low power leakage, and fast read speed. Embedded systems with hybrid memories can bring energy-efficiency and low latency to tele-health systems. In this paper, we investigated the efficiency of adopting hybrid memories in tele-health embedded systems. The proposed hybrid memories consist of ZRAM, SRAM, MRAM, and DRAM. Data allocation for the hybrid memory system is a major challenge. In this paper, we proposed dynamic programming and heuristic algorithms for data placement on hybrid memories. Dynamic programming can achieve efficient data allocation in polynomial computing time while heuristic approaches can reduce computing time and space. The experimental results have shown that we can significantly reduce the latency and energy cost compared to greedy algorithms.

9. Acknowledgment

This work is supported by NSF CNS-1457506 and NSF CNS-1359557.

References

[1] Medical devices & healthcare it, https://technology.ihs.com/researchareas/450450.

[2] S. Borkar, Thousand core chips: A technology perspective, in: ACM DAC, San Diego, CA, USA, 2007, pp. 746–749.

[3] M. Qiu, L. Chen, Y. Zhu, J. Hu, X. Qin, Online Data Allocation for Hybrid Memories on Embedded Tele-health Systems, in: High Performance Computing and Communications, 2014 IEEE 6th Intl Symp on Cyberspace Safety and Security, 2014 IEEE 11th Intl Conf on Embedded Software and Syst (HPCC,CSS,ICESS), 2014 IEEE Intl Conf on, 2014, pp. 574–579.

[4] M. Qiu, M. Guo, M. Liu, C. J. Xue, L. T. Yang, E. H.-M. Sha, Loop scheduling and bank type assignment for heterogeneous multi-bank memory, Journal of Parallel and Distributed Computing (JPDC) 69 (6) (2009) 546–558.

[5] M. Qiu, L. Yang, Z. Shao, E. Sha, Dynamic and Leakage Energy Minimization With Soft Real-Time Loop Scheduling and Voltage Assignment, Very Large Scale Integration (VLSI) Systems, IEEE Transactions on 18 (3) (2010) 501–504.

[6] M. Qiu, C. Xue, Z. Shao, E. Sha, Energy Minimization with Soft Real-time and DVS for Uniprocessor and Multiprocessor Embedded Systems, in: Design, Automation Test in Europe Conference Exhibition, 2007. DATE '07, 2007, pp. 1–6.

[7] C. Molina, C. Aliagas, M. Garcia, A. Gonzcalezo, J. Tubella, Non redundant data cache, in: Low Power Electronics and Design, 2003. ISLPED '03. Proceedings of the 2003 International Symposium on, 2003, pp. 274–277.

[8] X. Dong, X. Wu, G. Sun, Y. Xie, H. Li, Y. Chen, Circuit and microarchitecture evaluation of 3D stacking magnetic RAM (MRAM) as a universal memory replacement, in: Design Automation Conference, 2008. DAC 2008. 45th ACM/IEEE, IEEE, 2008, pp. 554–559.

[9] X. Guo, E. Ipek, T. Soyata, Resistive computation: Avoiding the power wall with low-leakage, STT-MRAM based computing, in: ISCA, 2010, pp. 371–382.

[10] M. Qiu, E. H.-M. Sha, Cost minimization while satisfying hard/soft timing constraints for heterogeneous embedded systems, ACM Transactions on Design Automation of Electronic Systems (TODAES) 14 (2) (2009) 1–30.

[11] P. Zhou, B. Zhao, J. Yang, Y. Zhang, Energy reduction for STT-RAM using early write termination, in: IEEE ICCAD, 2009, pp. 264–268.

[12] J.-G. Zhu, Magnetoresistive Random Access Memory: The Path to Competitiveness and Scalability, Proceedings of the IEEE 96 (11) (2008) 1786–1798.

[13] S. Okhonin, M. Nagoga, C.-W. Lee, J.-P. Colinge, A. Afzalian, R. Yan, N. D. Akhavan, W. Xiong, V. Sverdlov, S. Selberherr, et al., Ultra-scaled Z-RAM cell, in: Proc. IEEE Int. SOI Conf, 2008, pp. 157–158.

[14] A. SHANAVAS, Zero Capacitor RAM, http://www.edutalks.org/downloads/zram.pdf.

[15] J. Meza, J. Chang, H. Yoon, O. Mutlu, P. Ranganathan, Enabling efficient and scalable hybrid memories using fine-granularity DRAM cache managemen, IEEE Computer Architecture Letter 11 (2) (2012) 61–64.

[16] W. Tian, Y. Zhao, L. Shi, Q. Li, J. Li, C. Xue, M. Li, E. Chen, Task Allocation on Nonvolatile-Memory-Based Hybrid Main Memory, Very Large Scale Integration (VLSI) Systems, IEEE Transactions on 21 (7) (2013) 1271–1284.

[17] L. Zhang, M. Qiu, W. Tseng, E. Sha, Variable partitioning and scheduling for mpsoc with virtually shared scratch pad memory, J. Signal Process. Syst 58 (2) (2010) 247–265.

[18] Z. Chen, M. Qiu, SPM-aware Scheduling for Nested Loops in CMP Systems, SIGBED Rev. 10 (2) (2013) 13–13.

[19] S. J. E. Wilton, N. P. Jouppi, CACTI: An enhanced cache access and cycle time model, IEEE J. Solid-State Circuits 31 (5) (1996) 677–688.

[20] M. R. Guthaus, J. S. Ringenberg, D. Ernst, T. M. Austin, T. Mudge, R. B. Brown, MiBench: A free, commercially representative embedded benchmark suite, in: IEEE WWC, 2001, pp. 3–14.

[21] H. P. Wong, S. Raoux, S. Kim, J. Liang, J. P. Reifenberg, B. Rajendran, M. Asheghi, K. E. Goodson, Phase change memory, Proceedings of the IEEE 98 (12) (2010) 2201–2227.

[22] S. Lai, Current status of the phase change memory and its future, in: Electron Devices Meeting, 2003. IEDM'03 Technical Digest. IEEE International, 2003.

[23] M. Qiu, Z. Ming, J. Li, K. Gai, Z. Zong, Phase-Change Memory Optimization for Green Cloud with Genetic Algorithm, Computers, IEEE Transactions on 64 (12) (2015) 3528–3540.

[24] L. E. Ramos, E. Gorbatov, R. Bianchini, Page Placement in Hybrid Memory Systems, in: Proceedings of the International Conference on Supercomputing, 2011, pp. 85–95.

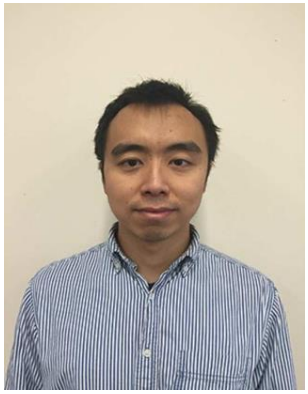
[25] J. Li, Z. Ming, M. Qiu, G. Quan, X. Qin, T. Chen, Resource allocation robustness in multi-core embedded systems with inaccurate information, J. Syst. Archit. 57 (9) (2011) 840–849.

[26] M. K. Qureshi, V. Srinivasan, J. A. Rivers, Scalable High Performance Main Memory System Using Phase-change Memory Technology, in: Proceedings of the 36th Annual International Symposium on Computer Architecture, ISCA '09, 2009, pp. 24–33.

[27] M. Qiu, Z. Ming, J. Li, S. Liu, B. Wang, Z. Lu, Three-phase Time-aware Energy Minimization with DVFS and Unrolling for Chip Multiprocessors, J. Syst. Archit. 58 (10) (2012) 439–445.

[28] M. Qiu, Z. Chen, Z. Ming, X. Qin, J. Niu, Energy-Aware Data Allocation With Hybrid Memory for Mobile Cloud Systems, Systems Journal, IEEE PP (99) (2014) 1–10.

[29] J. Li, M. Qiu, J.-W. Niu, L. T. Yang, Y. Zhu, Z. Ming, Thermal-aware Task Scheduling in 3D Chip Multiprocessor with Real-time Constrained Workloads, ACM Trans. Embed. Comput. Syst. 12 (2).



Longbin Chen Bio

Longbin Chen is a 1st year Ph.D. student of Computer Science at Pace University. He received his Bachelor's from Xiamen University and Master's degree from San Jose State University. His research interests include embedded systems, distributed systems, and cloud computing.

Meikang Qiu Bio

Meikang Qiu is an Associate Professor of Computer Science at Pace University. He received Ph.D. degree of Computer Science from University of Texas at Dallas. He is an IEEE Senior member and ACM Senior member. He has published 5 books, 300 peer-reviewed journal and conference papers (including 140+ journal articles, 160+ conference papers, 40+ IEEE/ACM Transactions papers), and 3 patents. He has won ACM Transactions on Design Automation of Electrical Systems (TODAES) 2011 Best Paper Award. His paper about cloud computing has been published in JPDC (Journal of Parallel and Distributed Computing, Elsevier) and ranked #1 in Top Hottest 25 Papers of JPDC 2012. He has won another 8 Conference Best Paper Awards in recent years. Currently he is an associate editor of 10+ international journals, including IEEE Transactions on Computer and IEEE Transactions on Cloud Computing.

Wenyun Dai Bio

Wenyun Dai is a 1st year Ph.D. student of Computer Science at Pace University. He received his Bachelor's Degree from Xiamen University and Master's Degree from Shanghai JiaoTong University. He has published several conference/journal papers on cloud computing and security.

Houcine Hassan Bio

Houcine Hassan received the M.S. and Ph.D. degrees in computer engineering from the UPV in 1993 and 2001, respectively. He is an Associate Professor with the Computer Engineering Department at the UPV, since 1994. He has published more than 100 papers in international refereed conferences and journals, and has been a scientific committee member of about 150 IEEE and ACM International Conferences and Journal Editorial Boards. His research interests focus on real-time systems, computer architectures, embedded systems, power savings, and industrial informatics.