Proposition for a Sequential Accelerator in Future General-Purpose Manycore Processors and the Problem of Migration-Induced Cache Misses
Pierre Michaud, Yiannakis Sazeides, André Seznec

To cite this version:

HAL Id: inria-00471410
https://hal.inria.fr/inria-00471410v2
Submitted on 8 Apr 2010

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L’archive ouverte pluridisciplinaire HAL, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d’enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.
Proposition for a Sequential Accelerator in Future General-Purpose Manycore Processors and the Problem of Migration-Induced Cache Misses

Pierre Michaud  
INRIA Rennes - Bretagne Atlantique  
Campus de Beaulieu  
35042 Rennes Cedex, France  
pierre.michaud@inria.fr

Yiannakis Sazeides  
Department of Computer Science, University of Cyprus  
75 Kallipoleos Street  
1678 Nicosia, Cyprus  
yanos@cs.ucy.ac.cy

André Seznec  
INRIA Rennes - Bretagne Atlantique  
Campus de Beaulieu  
35042 Rennes Cedex, France  
andre.seznec@inria.fr

ABSTRACT
As the number of transistors on a chip doubles with every technology generation, the number of on-chip cores also increases rapidly, making possible in a foreseeable future to design processors featuring hundreds of general-purpose cores. However, though a large number of cores speeds up parallel code sections, Amdahl’s law requires speeding up sequential sections too. We argue that it will become possible to dedicate a substantial fraction of the chip area and power budget to achieve high sequential performance. Current general-purpose processors contain a handful of cores designed to be continuously active and run in parallel. This leads to power and thermal constraints that limit the core’s performance. We propose removing these constraints with a sequential accelerator (SACC). A SACC consists of several cores designed for ultimate sequential performance. These cores cannot run continuously. A single core is active at any time, the rest of the cores are inactive and power-gated. We migrate the execution periodically to another core to spread heat generation uniformly over the whole SACC area, thus addressing the temperature issue. The SACC will be viable only if it yields significant sequential performance. Migration-induced cache misses may limit performance gains. We propose some solutions to mitigate this problem. We also investigate a migration method using thermal sensors, such that the migration interval depends on the ambient temperature and the migration penalty is negligible under normal thermal conditions.

1. INTRODUCTION
The continued doubling of transistor density with every technology generation has been leading to a rapid increase in the number of on-chip cores. So far, general purpose multicore processors are symmetric, i.e., all cores are identical. But future manycores will likely be asymmetric, with some cores specialized for certain tasks. In particular, since performance on sequential applications and on sequential sections in parallel applications will remain a major issue, future manycores may feature some cores for providing high sequential performance.

On current multicores, the power and temperature walls forbid aggressive design and very high clock frequencies. However, as the number of on-chip cores increases, each core will consume a smaller fraction of the total chip power and area. For example if, in ten years from now, technology affords the silicon area and the power budget for 1000 EV6-like cores on the same chip, then it might be performance effective to implement only 900 of those cores and to dedicate the rest of the silicon area and power budget to any microarchitecture providing very high sequential performance.

In this paper, we propose a new direction for utilizing a large fraction of the chip area and power budget to achieve high sequential performance. We refer to our approach as sequential accelerator, or SACC. A SACC consists of several large power-hungry cores (LPH) designed for ultimate instantaneous sequential performance by using aggressive microarchitecture, circuits and technology, high voltage and clock frequency. In the design of the LPH, the fundamental thermal constraint that a core should be able to run continuously is removed. Only a single LPH is active at any time. The inactive LPHs are power-gated. The overall SACC power can be directed to the active LPH. Having several cores makes possible to spread the heat generation on the whole SACC area by migrating the execution periodically to another LPH, thus solving the temperature issue.

Activity migration for temperature has already been considered in the framework of small scale multicores [33, 13, 30]. Our thesis is that sacrificing the possibility of parallel execution in the SACC will allow us to specifically design an LPH able to consume a large instantaneous power, which will facilitate unprecedented levels of instantaneous sequential performance. We will also point out that migrating activity from one LPH to another LPH in the SACC is simpler than in a conventional multicore.

The detailed implementation of the LPH is beyond the scope of this paper. We mostly focused on the global SACC architecture and on thermal aspects. In particular, even if the LPH achieves very high instantaneous performance, the SACC approach will be viable only if the SACC achieves high long-range performance. Migrations, especially migration-
induced cache misses, may incur a significant performance penalty. We propose some solutions for alleviating this impact. In particular, since only a single LPH is active at a time, we propose to use a write-back level-1 (L1) data cache and a write-through level-2 (L2) cache in each LPH. We show that the number of migration-induced L2 misses can be reduced by warming-up the L2 of next-to-be-active LPHs, which increases performance significantly when implementing a sensor-less migration scheme with a fixed migration interval. We also propose a migration method using thermal sensors that adapts the migration interval to the thermal conditions, while ensuring it cannot be shorter than a predefined value. The migration interval can vary with the ambient temperature, and we show that the performance loss due to migrations is negligible as long as the ambient temperature does not exceed the nominal value.

The paper is organized as follows. In Section 2, we argue that future manycores may use a substantial fraction of the chip area to implement one very large unit providing high sequential performance. We describe in this section the sequential accelerator, our proposition for such unit. Related work is mentioned in Section 3. In Section 4, we propose several schemes to mitigate the impact of migration-induced L2 misses. Our simulation results are presented and analyzed in Section 5. Finally, Section 6 concludes this study.

2. A CASE FOR SEQUENTIAL ACCELERATORS

2.1 Implications of Amdahl’s law in the many-core era

Some researchers have advocated for asymmetric many-cores featuring many “small & slow” (SS) cores and a few (e.g., a single) “big & fast” (BF) cores [23, 2, 1, 28, 14, 36]. Hill and Marty made the following fundamental observation [14]: “Increasing core performance, even if it appears locally inefficient, can be globally efficient by reducing the idle time of the rest of the chip’s resources.” In particular, Hill and Marty exhibit some examples of asymmetric manycores whose optimal configuration features one BF core using several hundred times the silicon area of one SS core. To corroborate their finding, we define a simple model where all area numbers are normalized to the area of a 256K L2 cache built in the same technology. More precisely, we consider the 256K L2 cache of the Intel Pentium III Coppermine, whose area was approximately 31 mm$^2$ in 180 nm technology. For information, Table 1 gives the approximate normalized areas of some known processing cores. \footnote{We obtained areas from die photos and from published die areas, and we assume that areas are divided by two on each technology generation.}

<table>
<thead>
<tr>
<th>core</th>
<th>core caches</th>
<th>core area</th>
</tr>
</thead>
<tbody>
<tr>
<td>IBM Cell SPE</td>
<td>256K local store</td>
<td>1.9</td>
</tr>
<tr>
<td>DEC Alpha 21064 (EV4)</td>
<td>16K+16K</td>
<td>0.7</td>
</tr>
<tr>
<td>DEC Alpha 21164 (EV5)</td>
<td>16K+8K</td>
<td>1.1</td>
</tr>
<tr>
<td>DEC Alpha 21264 (EV6)</td>
<td>64K+64K</td>
<td>2.4</td>
</tr>
<tr>
<td>Intel Pentium III</td>
<td>16K+16K+256K</td>
<td>3.4</td>
</tr>
<tr>
<td>Intel Atom</td>
<td>32K+32K</td>
<td>3.5</td>
</tr>
<tr>
<td>Intel Core 2</td>
<td>32K+32K</td>
<td>8.1</td>
</tr>
<tr>
<td>Intel i7 (Nehalem)</td>
<td>32K+32K+256K</td>
<td>14</td>
</tr>
</tbody>
</table>

![Figure 1: Speedup as a function of the BF core area $b$ for a parallelizable fraction $f = 0.9$ and $f = 0.99$ and a small core area $s = 1, 4, 16$. The core performance vs. area model is assumed to be $p(x) = x^{1/4}$.

Hill and Marty modeled performance as $p(x) = \sqrt{x}$, following Pollack’s rule [3]. We use the more conservative model $p(x) = x^{1/4}$. Figure 1 shows the speedup as a function of $b$ for different values of $f$ and $s$. Notice that $p(x) = x^{1/4}$ corresponds to a locally quite inefficient use of silicon area (doubling sequential performance requires 16 times larger area). Yet, on these examples, the optimal BF core area is fixed...
Computer architects define new general-purpose microarchitectures several technology generations ahead, at a time when the characteristics of the applications that will be running on that microarchitecture are not known. Considering the current state of the software ecosystem, dimensioning a general-purpose manycore on the assumption that all the applications running on that manycore will be embarrassingly parallel would be overly optimistic. Without knowing the actual value of \( f \), it is a safe choice to use about half of the silicon area for the BF core. On our example, this means a BF core normalized area of 528, i.e., about 38 times the silicon area for the BF core. On our example, this means a BF core normalized area of 528, i.e., about 38 times the area of an Intel i7 core (Table 1). The sequential accelerator we present in the following is our proposition for such BF logical core.

### 2.2 The sequential accelerator

So far, we assumed that it will be possible to build a BF core several tens of times bigger than the biggest cores existing today, yet delivering higher performance. A simple way to make bigger cores is to enlarge core cache, which benefits some applications. But once the working set fits in, enlarging a cache further brings no performance gain. It may even decrease performance because of the longer cache access time. Another possible way to make bigger cores would be to exploit more instruction-level parallelism (ILP). ILP techniques that were deemed too power hungry in the past could become viable in the manycore era. Yet, ILP alone is unlikely to suffice. A new approach is required.

Our proposition is to implement one BF logical core consisting of several large power hungry (LPH) physical cores, where a single LPH can be active at any time, and the inactive LPHs are power gated [37, 22]. We assume that the LPH circuit and technology parameters will be specialized for high clock frequencies, which will yield high sequential performance at the cost of a high power density. In order to tolerate a high power density and keep temperature below an acceptable value, when needed, the execution is migrated to another LPH [13, 30]. We call such BF logical core a sequential accelerator (SACC). Figure 2 shows an example of SACC consisting of 16 LPHs on one fourth of the total chip area. If the power density in the inactive LPHs can be kept low, the majority of the power budget allocated to the whole SACC can be concentrated in one LPH at a given time, which will make possible a very aggressive LPH design. Actually, the power budget instantaneously available for one LPH increases with the number of valid LPHs \(^3\) in the SACC. There are several possible ways to use this large power budget and increase sequential performance. A possible way is to overclock the LPH by increasing the supply voltage \( V_{dd} \). For instance, the IBM POWER6 clock frequency can increase from 4 GHz to 6 GHz by rising \( V_{dd} \) from 0.9V to 1.3V [35]. Similar trends were observed for the IBM Cell [29]. Increasing \( V_{dd} \) is not the only way to increase the clock frequency. The LPH microarchitecture can be specifically designed for high clock frequency by implementing long pipelines and by shortening critical paths, using latches and flip-flops optimized for speed, dynamic logic, low-Vt transistors, low-voltage swing logic [7], etc. ILP techniques will be important too, especially for latency tolerance (large instruction window, complex branch predictor, complex cache prefetcher, etc.). Several questions must be answered, among which:

1. How much power can the supply grid deliver to the active LPH?
2. How effective will power-gating techniques be?
3. What will be the microarchitecture of the LPH, and how much sequential performance will be gained through aggressive yet reliable design?
4. How frequently the execution should be migrated, and should we implement special support to decrease the migration penalty?

The first two questions, and to some extent the third one, will be tackled at the technology and circuit levels. We leave these questions for future studies. In the following, we focus on the last question.

### 3. RELATED WORK

#### 3.1 Increasing sequential performance

Many methods for exploiting more instruction-level parallelism (ILP) have been proposed by researchers in the last two decades. These include methods for making the instruction pipeline wider, methods for enlarging the instruction window, better branch predictors, cache prefetch mechanisms, etc. Beyond ILP, researchers have explored speculative multithreading, where several processing units or cores are used to accelerate programs that cannot be parallelized by conventional means [34, 21, 12]. More recently, some researchers have proposed to aggregate several narrow-issue physical cores to form a wide-issue core [17, 19]. Over-clocking is another method for increasing sequential performance. On the Intel i7 processor, when some cores are inactive and in low-power mode, the frequency and voltage of the active cores increases automatically [15]. The BubbleWrap manycore proposition is also based on overclocking.

---

\(^3\) Because of transistor variability, there may be some malfunctioning LPHs detected at manufacturing time. The SACC control must be programmed at manufacturing time so that invalid LPHs are removed from the migration path. When this happens, the effective SACC area for dissipating heat is reduced, which requires lowering the power envelope hence voltage and clock frequency.

\(^4\) For instance, the IBM POWER6 and Cell processors have been designed to have a short clock cycle in FO4 delays, and they use high-speed circuit techniques [35, 29], whereas the Intel i7 core was optimized for performance per watt [22].
ing [18]. To make overclocking more effective, the BubbleWrap manycore features some expendable cores whose lifetime is relatively short because of the high voltage and high temperature, and expendable cores are consumed one by one. Constant-voltage overclocking is possible using a leader-checker core pair where the leader core is overclocked and the checker core detects and repairs errors [10]. It was shown recently that the benefits of conventional overclocking and constant-voltage overclocking are additive and that combining them yields substantial performance gains [9].

The SACC differs from previous propositions in that LPHs are specifically designed not to be active simultaneously, which should yield higher sequential performance that can be obtained by just overclocking standard cores.

### 3.2 Activity migration

The idea of having spare execution resources to continue the execution in case of a thermal emergency was proposed in [24, 33]. The idea of activity migration as a general way to decrease time-averaged power density was introduced and explained by Heo et al. [13]. In particular, they have emphasized the fact that higher power densities require smaller migration intervals. They also considered having a spare core identical to the main core. Spare cores are naturally present in chip multiprocessors when there are fewer running threads than cores. Some studies have considered activity migration in the context of chip multiprocessors running multiprogrammed workloads, with thread migration methods using thermal sensors [30, 8, 27, 5]. Constantinou et al. quantified the migration penalty for a single thread migrating periodically on a chip multiprocessor [6]. Moreover, in some of the schemes we propose in this section, some inactive LPHs may have their L2 powered and accessed. To ease the discussion, we distinguish logical and physical LPHs. The SACC has 16 physical LPHs denoted P(0) to P(15). The number of logical LPHs is unlimited. Logical LPHs are denoted L(n). The logical LPH L(n) is mapped onto the physical LPH P(n mod 16). The migration path is as follows: if the current active LPH is L(n), the next active LPH will be L(n + 1), i.e., the logical LPH number keeps increasing. This means a circular migration path on physical LPHs. Henceforth, we assume that the L1 data cache uses a write-back policy because this reduces power consumption. Unless specified otherwise, the L2 is write-back. Before migrating from L(n) to L(n + 1), we flush the L1 data cache of L(n). That is, dirty L1 blocks on L(n) are written back in the L2 (or directly in the L3 if the block is not in the L2 or if the L2 is write-through). As the L1 cache is small, this flushing can be done relatively quickly and the impact on the migration penalty is small. Moreover, we assume that the L2 of L(n + 1) is initially empty.

#### 4.1 Flush-before-migration (FBM)

With a write-back L2, the simplest migration method flushes the L2 of L(n) before migrating to L(n + 1). That is, dirty blocks on L(n) are written back in the L3. Once the flushing is done, the L2 of L(n) can be turned off and the execution can resume on L(n + 1). Upon an L2 miss, the miss request is fulfilled by the L3. We call this scheme flush-before-migration (FBM).

#### 4.2 Flush-after-migration (FAM)

With a write-back L2, another possibility is to migrate the execution to L(n + 1) without waiting for the L2 of L(n) to be flushed, that is, the flushing is done in background. In this case, some remote L2s (i.e., L2s of inactive LPHs) may be powered and holding valid blocks. Actually, it is possible that the only valid copy of a cache block lie in a remote L2. Upon a miss in its local L2, the active LPH accesses simultaneously the L3 and the remote L2s. In case of a hit in a remote L2, the block from the L3 is dropped. To simplify the hardware, we maintain an invariant which is that at most one LPH can have a copy of a block. This way, we avoid having to select between several hitting L2s. To maintain this invariant, when a L2 miss request is fulfilled by a remote L2, the block copy on the remote L2 is invali-
dated. We call this a block migration. For a block migration on write-back L2s, the dirty bit of the remote block is propagated along with the block and its value is preserved when storing the block in the L2 of the active LPH. It should be noted that the invariant guarantees that all the blocks on inactive LPHs are up-to-date. Indeed, if a block is on an inactive LPH, it means that it has not been requested by the active LPH, so it has not been modified. Depending on how aggressively we want to save power, we may start flushing the L2 as soon as possible, or with some delay. For instance, we may start flushing the L2 of \( L(n) \), block by block, just after having migrated the execution from \( L(n) \) to \( L(n+1) \). In this case, the shared-bus bandwidth is time-shared between the LPHs using a fair arbitration. We call this scheme flush-after-migration (FAM). Once the L2 of \( L(n) \) is completely flushed, it can be turned off. 5

If we want to take advantage of the smaller latency of L2-to-L2 transfers compared with L3-to-L2 ones, we may choose to start flushing the L2 of \( L(n) \) only when migrating from \( L(n+k) \) to \( L(n+k+1) \). This is a generalization of flush-after-migration, which we denote FAM-k (FAM-0 is the same as FAM).

### 4.3 L2 warm-up (WU)

A possible way to decrease the number of migration induced L2 misses is to warm up the L2 of LPHs on which we are going to migrate next. Upon an L2 miss on the active LPH \( L(n) \), the missing block can be snooped on the shared bus by LPHs \( L(n+1) \) to \( L(n+j) \) and stored in their L2s. This requires to power up the L2 of \( L(n+j+1) \) when migrating from \( L(n) \) to \( L(n+1) \). The \( j \) LPHs \( L(n+1) \) to \( L(n+j) \) are called the future LPHs. We denote such scheme WU-J. L2 warm-up can be combined easily with FBM: when a dirty block is evicted from the L2 and written back to the L3, future LPHs snoop the bus and update their block copy. However, combining L2 warm-up with FAM is more complex. Upon an L2 miss on the active LPH \( L(n) \), we access simultaneously the L3 and the L2s of past LPHs \( L(i) \), \( i < n \). But there are some complications that must be addressed. If we update future LPHs only on L2 write-backs, L2 hits on blocks resulting from warm-up may return stale data. Also, on a L2 miss, there may be several hitting past LPHs. Moreover, blocks on past LPHs may not be up-to-date.

Actually, these problems can be solved with a write-through L2. A write-through L2 generates more write traffic on the shared bus, but this effect is somewhat mitigated by the L1 write-back policy. When a dirty block is evicted from the L1 data cache, the block is written to the L3, and also in the L2 if the L2 has an older copy. Past and future LPHs snoop the bus and update their copy of the block. In case of a L2 miss, and if there are some hits on several past LPHs, we select among hitting past LPHs \( L(i) \) the one with the largest \( i \), i.e., the one that has been active most recently. It should be recalled that we flush the L1 data cache prior to migrating. It should also be noted that the L2 of \( L(n) \) can now be turned off instantaneously when migrating from \( L(n+k) \) to \( L(n+k+1) \). We denote such scheme using a write-through L2 Lag-k, to distinguish it from FAM-k.

Schemes characteristics are summarized in Table 2. Note that Lag-k WU-0 schemes use block migration, so we have at most a single hit on past LPHs for these schemes.

### 5. SIMULATIONS

Our simulator is trace driven. We model the LPH approximately: when there are no cache misses and no branch mispredictions, the LPH executes two x86 instructions per cycle. 6 Instructions are fetched and retired in program order. Loads and stores are pipelined and non-blocking. We model caches, request queues, and bandwidth contention. Upon a branch misprediction, we wait until the instruction window is completely drained before resuming instruction fetching. The branch predictor is never turned off and keeps its information even when the LPH is inactive. The main parameters of the simulated microarchitecture are listed in Table 3. We assume that the shared bus is clocked at a low frequency.

---

5This is a rough approximation, but our qualitative conclusions are largely independent of a precise ILP value.
negligible compared with the migration intervals considered to transfer the architectural register values, as this time is immediately on the new active LPH except for FBM and FBM WU schemes. We did not model the time necessary to resume the execution is completely drained. Then we start flushing the L1 data cache. Once the L1 data flushing is finished and the L1-to-L2 write-back queue is drained, we resume the execution immediately on the new active LPH except for FBM and FBM WU schemes. We did not model the time necessary to transfer the architectural register values, as this time is negligible compared with the migration intervals considered in this study. When LPHs compete for the shared-bus bandwidth (FAM-k schemes), we simulate an arbitration mechanism that allots bandwidth equally among competing LPHs. Traces were generated with Pin [25] for the SPEC CPU 2006 benchmarks. For each benchmark, we skip the first 30 billion instructions, and the trace represents the next 1 billion instructions.

5.1 Sensor-less migrations

In this section, we consider the case where the migration interval is fixed, i.e., the execution is forced to migrate at regular intervals whatever the ambient thermal conditions. This solution is simple to implement and does not rely on thermal sensors. However, the migration interval value must be fixed carefully. Temperature on a given LPH increases when this LPH is active and decreases when it is inactive. Consequently, temperature on the LPH oscillates around an average value which is a function of the total SACC power. The peak temperature, though, increases with the migration interval. If the migration interval is too long, temperature on the active LPH may exceed the limit. On the other hand, if the migration interval is too short, migrations incur a large performance penalty. Using ATMI [26] and assuming a power density of 8 W/mm², we found that the migration interval must be a few tens of microseconds in order to keep the amplitude of the temperature oscillation within a few degrees. Unless stated otherwise, we have assumed a migration interval of 25 µs, i.e., 200,000 clock cycles at 8 GHz.

The baseline performance is the performance of the baseline configuration (Table 3) when migration is disabled, i.e., when the execution keeps executing on the same LPH, ignoring temperature issues. We simulated 12 different schemes of varying complexity. The normalized performance, for a scheme X on a particular benchmark, is the baseline execution time divided by the execution time of X (the normalized performance rarely exceeds 1). Then, we compute the harmonic mean of the normalized performance over all benchmarks. Performance numbers are shown in Figure 3. We also show the performance of a core with the same characteristics as the baseline, but with the LPH clock frequency and the shared bus frequency both at 4 GHz. Figure 4 shows the arithmetic mean, on all benchmarks, of the number of instructions transiting through the shared bus per LPH cycle (8 GHz). This measures the bus activity. Everything else being equal, we want the bus activity to be as small as possible as it represents some dynamic power consumption, from the bus itself, from the L3, and from the remote L2s that

\[ \text{normalized performance} \]

\[ \text{migration interval (cycles)} \]

\[ \text{normalized performance} \]

\[ \text{migration interval (cycles)} \]
are accessed (reads on past LPHs, writes on future LPHs). Note that L2 warm-up is very effective: not only does it increase performance, but it also decreases bus activity, despite a write-through L2. We conclude from these results that Lag-0 WU-j schemes are the most attractive, for the reasons explained below.

From our results, FBM WU-j is not competitive compared with Lag-0 WU-j. Indeed, the performance of Lag-0 WU-j is higher than that of FBM WU-j. The saving on the shared-bus traffic thanks to the write-back L2 is not that large. The problem with the write-back L2 is that the limited shared-bus bandwidth sometimes makes the L2 flush time relatively long compared with the migration interval, hence there is a large migration penalty. The performance of FBM ranges from 0.57 to 0.91. On some benchmarks, it is even outperformed by the 4 GHz core. Even though the write-back L2 generates less bus traffic, this traffic is more bursty. The problem could be solved by clocking the bus at a higher frequency, but this would make the implementation more complex and would increase the power consumption, possibly making the shared bus a thermal hot spot.

The performance of FAM-k is higher than that of FBM. The performance gain comes both from having to wait for L2 flushing before migrating (FAM-0) and from the smaller latency of L2-to-L2 misses compared to L3-to-L2 ones. However, the performance of FAM-2 is barely better than that of FAM-1, because fewer benchmarks have blocks with a reuse distance between 200k and 400k cycles. Nevertheless, FAM-k does not look like an interesting design point. Indeed, FAM-k is more complex than Lag-0 WU-k (cf. Table 2). Moreover, its performance is slightly less than that of Lag-0 WU-k, and FAM-1 and FAM-2 schemes access the shared bus more often than Lag-0 WU-1 and Lag-0 WU-2. Lag-k WU-0 is less attractive than Lag-0 WU-k, because it is slightly more complex to implement, because its performance is slightly lower, and because it generates more bus activity (Figure 4). As for Lag-1 WU-1, it is not an interesting design point, because Lag-0 WU-2 offers the same performance and is simpler to implement (cf. Table 2). Moreover, Lag-0 WU-2 generates less bus activity than Lag-1 WU-1.

Figure 5 shows the harmonic mean over all benchmarks of the normalized performance as a function of the migration interval. L2 warm-up improves performance significantly, even for relatively long migration intervals. For instance, with an interval of 500k cycles, Lag-0 WU-2 is on average 5% more performant than Lag-0 WU-0 (18% on 473.astar, 16% on 435.gromacs). The migration interval and the degree of L2 warm-up should be programmable and set at manufacturing time based on the number of valid LPHs and on the LPH power consumption.

In the next section, which presents a sensor-based migration method, we consider only Lag-0 WU-0 and Lag-0 WU-1 schemes. Lag-0 WU-0 consumes on average more power than Lag-0 WU-1 in the shared bus (Figure 4). But Lag-0 WU-1 consumes more power than Lag-0 WU-0 in the L2. Overall, it is not clear whether one scheme will consume significantly more power than the other. In our simulations, we assumed that both schemes consume the same power.

### 5.2 Sensor-triggered migrations

Sensor-less migration keeps the migration interval fixed even under mild thermal conditions. For instance, when the ambient temperature is low, or when running a “cold” application, it may be possible to have a longer migration interval without exceeding the temperature limit. This would decrease the performance loss due to migrations. If we trigger migrations based on thermal sensor information instead of having a fixed migration interval, we can adjust the migration interval dynamically depending on thermal conditions.

For our simulations, we assume that each LPH features a temperature sensor located at the LPH center. A sensor gives a temperature measurement every 2.5 μs (i.e., every 20,000 LPH cycles at 8 GHz), which we call a sensor cycle. A simple yet inapplicable strategy would be to trigger a migration as soon as temperature on the active LPH exceeds the temperature limit $T_{max}$. Such strategy is inapplicable because the migration interval may become extremely short when the average temperature in LPHs is close to $T_{max}$. Actually, activity migration does not preclude the necessity of throttling power consumption under higher-than-nominal thermal conditions. The solution we propose relies on a linear on/off throttling mechanism (aka stop-go) similar to the one used in the Intel Pentium 4 [11]. More precisely, we propose the following method. As long as temperature stays below $T_{max}$, we keep executing on the same LPH. When temperature exceeds $T_{max}$, and if the time $t - t_{mig}$ elapsed since the last migration is longer than a fixed value $t_{min}$, the execution migrates to a another LPH. When temperature exceeds $T_{max}$ but $t < t_{mig} + t_{min}$, a migration is scheduled to happen at $t_{mig} + t_{min}$, but in the meantime the active LPH enters a low-power state for one sensor cycle (and possibly several sensor cycles if temperature exceeds $T_{max}$ several times between $t_{mig}$ and $t_{mig} + t_{min}$). This method forces the migration interval to be longer than $t_{min}$.

### Simulation hypotheses.

We used the migration path corresponding to the LPH numbering shown in Figure 2. However, as far as temperature is concerned, the impact of the migration path is negligible. With a shared bus, this particular migration path is as good as any other. We assume that the SACC is designed for an ambient temperature not exceeding a nominal value of 40°C, i.e., we have chosen the nominal power and heatsink thermal resistance so that, when the migration interval is short enough, the average temperature in LPHs is approximately 90 °C with a local ambient at 40 °C. For simulating temperature, we used the ATMI model [26]. Our simulation parameters are listed in Table 4. We modeled LPHs as squares with a uniform power density. The

---

Table 4: Thermal and physical parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>heatsink thermal resistance</td>
<td>0.4 K/W</td>
</tr>
<tr>
<td>copper plate</td>
<td>7 cm x 7 cm x 5 mm</td>
</tr>
<tr>
<td>silicon thickness</td>
<td>500 μm</td>
</tr>
<tr>
<td>interface material</td>
<td>100 μm, 3 W/(mK)</td>
</tr>
<tr>
<td>total SS cores area</td>
<td>64 mm²</td>
</tr>
<tr>
<td>number of LPHs</td>
<td>16</td>
</tr>
<tr>
<td>LPH size</td>
<td>2 mm x 2 mm</td>
</tr>
<tr>
<td>local ambient $T_{amb}$</td>
<td>variable</td>
</tr>
<tr>
<td>temperature limit $T_{max}$</td>
<td>90 °C</td>
</tr>
<tr>
<td>sensor cycle</td>
<td>20000 LPH cycles</td>
</tr>
<tr>
<td>SS cores power density</td>
<td>≤ 0.5 W/mm²</td>
</tr>
<tr>
<td>LPH power density</td>
<td>active on: 8 W/mm²,</td>
</tr>
<tr>
<td></td>
<td>inactive: 0.8 W/mm²</td>
</tr>
</tbody>
</table>

---

The migration path might impact inductive noise.

The local ambient temperature is the temperature inside the computer case, at the CPU fan inlet. It is typically several degrees Celsius above the room temperature [16].
nominal power density in the SS cores is 0.5 W/mm² (when necessary the SS cores are throttled, with the same throttling factor). The nominal power allocated to the SACC is equal to that allocated to the SS cores, i.e., 32 W. We assume that this power is concentrated in the active LPH, which means a power density of 8 W/mm². Note that if we disable migration and keep executing on the same LPH at nominal ambient, temperature in the active LPH exceeds 170 °C. We assumed that the drowsy state of the branch predictor is effective and we neglected the power consumption of inactive LPHs. While in a low-power off state, the active LPH consumes some static power in the microarchitected tables (registers, caches, TLBs,...). Moreover, the sensor cycle is relatively short and power gating may not be effective instantaneously [37]. Consequently, we assumed that the power density of the active LPH in the off state is 1/10 the power density in the on state.

When simulating the impact of thermal throttling, the initial thermal state may have a large impact. Because our simulations are short, we want the initial thermal state to be as close as possible to the steady state. In the ATMI model, the initial thermal state is set indirectly: it is the steady-state corresponding to some specified power densities. To set the initial thermal state, we must determine approximately the power densities generated by long-term thermal throttling. We start from a uniform power density of 0.5 W/mm² in all cores (including SS cores, modeled as a single big core), and we compute the corresponding steady-state temperatures. If the hottest core is hotter than \( T_{\text{max}} \), we decrease the power density in that core by a tiny amount. We iterate this process (the hottest core may change as we iterate) until the steady-state temperature is less than or equal to \( T_{\text{max}} \) in all cores. We use the resulting power densities to set the initial thermal state.

**Performance depends on ambient temperature.**

With sensor-less migrations and under normal thermal conditions, performance does not depend on the ambient temperature. This is no longer the case with sensor-triggered migrations. The migration interval depends not only on the ambient temperature but also on the heatsink temperature, i.e., what applications have been running previously. Of course, we expect migrations to have little impact under normal thermal conditions. But it will be almost impossible to replay the same program and get exactly the same execution time. Note however that this is already very difficult to achieve on existing platforms.

**Results.**

Our simulation results are summarized in Figure 6 and Table 5. When \( T_{\text{amb}} = 40 \) °C, performance is within 10% of the baseline performance for most benchmarks, provided \( t_{\text{min}} \) does not exceed a few hundred thousand cycles. Lag-0 WU-1 is only 3% better than Lag-0 WU-0 on average because the migration interval is relatively long. It must be noted that \( t_{\text{min}} = 10^7 \) cycles gives a very low performance, even at \( T_{\text{amb}} = 40 \) °C. In this situation, the thermal limit is underexploited because the time till the execution revisits the same LPH again is long enough for temperature to drop significantly below \( T_{\text{max}} \). Hence the time-averaged temperature is significantly lower than \( T_{\text{max}} \). Time-averaged temperature relative to \( T_{\text{amb}} \) being proportional to power, the total power that can be dissipated is smaller for larger values of \( t_{\text{min}} \), and throttling triggers more often. As the ambient temperature increases above 40 °C, the migration interval tends to decrease and the difference between Lag-0 WU-0 and Lag-0 WU-1 becomes more significant. At \( T_{\text{amb}} = 50 \) °C and for \( t_{\text{min}} = 100k \), Lag-0 WU-1 is on average 7% faster than Lag-0 WU-0 (up to 24% on one benchmark). For very small values of \( t_{\text{min}} \) and \( T_{\text{amb}} \) above 40 °C, the migration interval is small, so the temperature oscillation on each LPH has a small amplitude and the time-averaged temperature is close to \( T_{\text{max}} \). However, in this case, it is the migration penalty that limits performance. Consequently, for a given value of \( T_{\text{amb}} \), there exists an optimal value for \( t_{\text{min}} \). For instance,
for $T_{amb} = 50$ °C, the optimal $t_{min}$ is around 100k cycles on average. Actually, if we look at benchmarks individually, the optimal $t_{min}$ is variable and depends on the migration penalty, which depends on application characteristics. Figure 7 shows normalized performance for Lag-0 WU-1 on a few benchmarks when $T_{amb} = 50$ °C. Setting $t_{min}$ to 100k cycles provides a good tradeoff overall, but it is sub-optimal on a per-application basis.

In summary, the SACC will deliver maximum performance as long as the ambient temperature does not exceed the nominal value. However, above the nominal ambient temperature, each extra degree Celsius will decrease performance significantly. This problem can be solved by using dynamic voltage/frequency scaling (DVFS). It is not clear to what extent it will be possible to use fine-grained DVFS in the future. The advantage of linear throttling is that it reacts quickly, which is important for a sensor-triggered migration scheme because the high power density in the active LPH makes temperature increase very fast. Nevertheless, coarse-grained DVFS can be combined with linear throttling, i.e., voltage and frequency are adjusted automatically, but slowly, to maintain the fraction of linear throttling below a threshold. This should be an effective approach, as the ambient and heatsink temperatures vary slowly in general.

Unlike the case of sensor-less migration, where Lag-0 WU-1 brings a clear performance benefit, Lag-0 WU-0 may be considered enough for sensor-triggered migrations. Lag-0 WU-1 brings a performance gain only when the ambient temperature is close to or higher than the nominal value.

5.3 Remarks about simulation hypotheses

We have assumed a 8 GHz clock frequency for the LPH in 11 nm technology. However, our qualitative conclusions are largely independent of a precise frequency value. We also ran some simulations assuming a 16 GHz clock frequency (with the bus still at 500 MHz). We observed a higher average performance, but our qualitative conclusions still hold.

We assumed that the power consumption of inactive LPHs is much smaller than the power consumption of the active LPH. We also assumed that the power grid could deliver several tens of watts to the active LPH. In practice, constraints may limit the power allocated to the active LPH. A smaller power budget for the active LPH means a less aggressive LPH design and/or a lower voltage/frequency setting. It also means a lower power density, hence longer migration intervals.

In our simulations, we assumed the same nominal power density for all the applications. Actually, power density depends on the application characteristics. The design, voltage and frequency setting of the LPH will be based on typical "hot" applications. With sensor-triggered migration, "cold" applications will experience longer migration intervals because of their lower power density.

6. CONCLUSION

In future general-purpose manycore processors, a large part of the chip area and power budget will likely be dedicated to high sequential performance. Microarchitecture and circuit design solutions that are deemed too power hungry today may help maximize the instantaneous sequential performance of a core, provided the core is not required to be continuously active. Based on this assumption, we proposed a sequential accelerator (SACC) consisting of several large power-hungry cores (LPHs), where a single LPH is active at a given time and other LPHs are power-gated. When needed, the execution is migrated to another LPH in order to spread the heat generation uniformly over the whole SACC area.

The effectiveness of a SACC may be significantly impaired by the migration penalty. We pointed out the potential impact of migration induced L2 misses on performance. We proposed and evaluated some schemes for decreasing this impact. We made a case for write-through L2 caches and write-back L1 data caches. We showed that warming-up the L2 of future LPHs on which we will migrate next is more interesting than fetching blocks from past LPHs. We showed that L2 warm-up may bring a significant performance gain for a sensor-less migration scheme with a fixed migration interval. We also defined a migration policy using thermal sensors which forces the migration interval to be longer than a predefined value. With this policy, the migration interval is variable and depends on the ambient temperature. As long as the ambient temperature stays below the nominal value, the migration interval is long and migrations incur little performance loss. However, when the ambient temperature exceeds the nominal value, performance degrades significantly if linear throttling is the only throttling method implemented.

Our study tackled only some of the questions that must be answered to make the sequential accelerator a viable solution. This work points to several new directions for future work, including microarchitectural design of LPHs, effective power-gating techniques, and effective power delivery to the active LPH.

7. ACKNOWLEDGMENTS

This work was partially supported by the European Commission in the context of the SARC integrated project #27648 (FP6) and by Intel.

8. REFERENCES


