Distributing the Frontend for Temperature Reduction

Pedro Chaparro, Grigorios Magklis, José González and Antonio González
Intel Barcelona Research Center - Intel Labs – UPC
{pedro.chaparro.monferrer, grigoriosx.magklis, pepe.gonzalez, antoniox.gonzalez}@intel.com

Abstract

Due to increasing power densities, both on-chip average and peak temperatures are fast becoming a serious bottleneck in processor design. This is due to the cost of removing the heat generated, and the performance impact of dealing with thermal emergencies. So far microarchitectural techniques to control temperature have mainly focused on the processor backend (in particular the execution units), whereas the frontend has not received much attention. However, as the temperature of the backend remains controlled and the processor throughput increases, the heat dissipated by the frontend becomes more significant, and one of the major contributors to the total average temperature.

This paper proposes and evaluates a distributed frontend for clustered microarchitectures that is able to reduce power density and temperature. First, a distributed mechanism for renaming and committing instructions is proposed. Second, a sub-banked trace cache with a bank hopping mechanism is presented. Finally, a method to improve the sub-banking is proposed based on a biased mapping function to distribute bank accesses to balance temperature.

1. Introduction

Power dissipation is one of the major hurdles in the design of next-generation microarchitectures. Power density is increasing in each generation due to the fact that frequency and leakage currents are scaling up very fast and their effect cannot be offset by decreasing the supply voltage. Power density directly translates into heat which must be removed from the processor die in order to keep the silicon temperature below a certain limit. The increase in power density makes the cost of the cooling system grow and challenges the performance benefits that can be obtained by the ever growing transistor density. For instance, traditionally the cooling system of a processor was designed to support the worst case peak temperature. Because of the growth of the cooling solution cost and some form factor constraints (especially in mobile computers), the cooling system is now designed for the common case and a thermal emergency mechanism is in charge of restoring the processor to its operating temperature. This solution has been adopted because the processor spends most of the time running at much lower temperatures than the worst-case scenario. Whenever a thermal emergency arises, a back-up mechanism to cool down the chip is triggered. Such mechanisms have a negative impact on performance.

The cost of the cooling system has been quantified in the order of \$1-3 or more per Watt when the average power exceeds 40 Watts [4][14], which represents a significant part of the total cost of the chip. The cost of the heat removal system is especially important for data centers where air conditioning is a main contributor over the whole data center cost [22]. Furthermore, circuit reliability depends exponentially upon operating temperature. Temperature variations account for over 50% of electronic failures [28].

In order to reduce dynamic power dissipation, chip designers rely on scaling down the supply voltage. To counteract the negative effect of a lower supply voltage on gate delay, the threshold voltage is also scaled down along with the supply voltage. However, lowering the threshold voltage has a significant impact on leakage current due to the exponential relationship between them. In fact, it is expected that within a few process generations the contribution of leakage power to the total power will be comparable to that of dynamic power [4][9]. It is also important to note that leakage power is exponentially dependent on temperature.

On the other hand, wire delays scale much slower than gate delays [1][3][21] and pose a serious obstacle to the scalability of superscalar processors. Clustered microarchitectures are an effective organization to deal with the problem of wire delays and complexity by means of partitioning some of the processor resources [6][11], such as the processor backend, and attempting to minimize the use of global (slow) communications.

Clustered microprocessors achieve a significant reduction of the backend temperature due to an effective distribution of the activity among the different clusters [8]. The frontend accounts for an important part of the processor area (about 39% in the Pentium® 4 at 90nm [29], and 20% in the clustered architecture presented in this paper) and power (at least 29% for the Pentium® Pro [20]; 30% of the dynamic power and 36% of the leakage for our micro-architecture). Figure 1 shows temperature results for our baseline clustered architecture averaged for the 26 SPEC2000. The frontend exhibits some of the highest temperatures in the processor (about 107ºC peak temperature and 70ºC average temperature) and, in our design, is also one of the largest contributors to temperature, which motivates our effort to reduce the temperature of the frontend. In any case, previous papers have already addressed the temperature problem in the clustered backend [7].

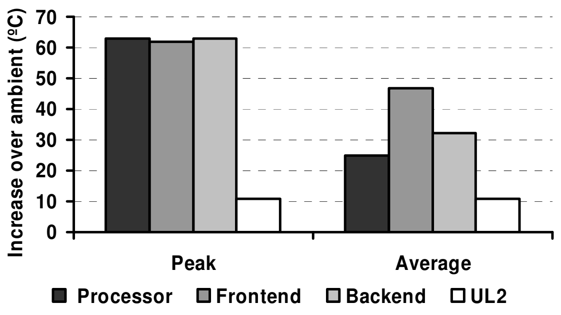


Figure 1. Temperature comparison of the different processor elements

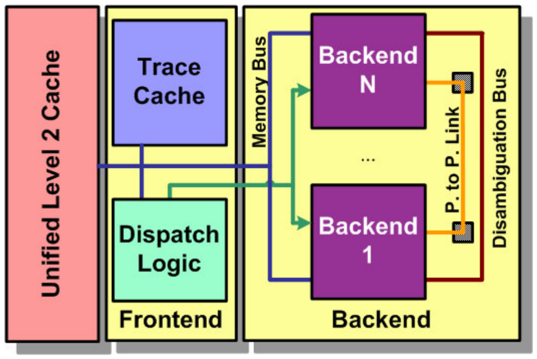
This paper proposes a distributed frontend for clustered microarchitectures with the goal of reducing on-chip temperature. Two different techniques are proposed: a distributed rename and commit mechanism and a bank hopping scheme for the trace cache with a thermal-aware mapping function.

The rest of the paper is organized as follows: Section 2 briefly describes the processor micro-architecture and the power and temperature models. Section 3 describes the proposal for the distributed rename and commit mechanism and the thermal-aware, sub-banked trace cache. Section 4 presents the performance results (execution time and temperature). Section 5 reviews the related work, and Section 6 concludes the paper.

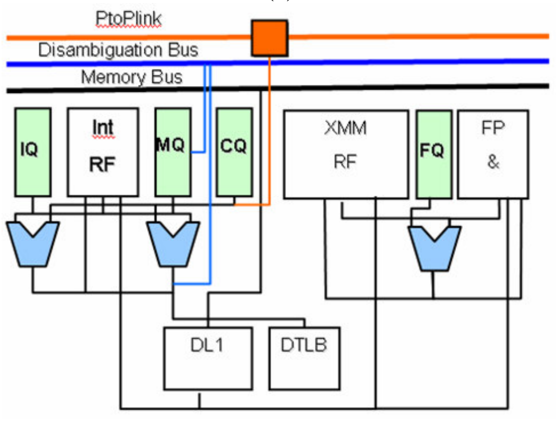
2. Processor Architecture

Figure 2 depicts the block diagram of the assumed clustered microarchitecture. Figure 2a shows the two main components of the processor: the frontend and the clustered backends. The frontend reads IA32 instructions from the UL2, translates them into *micro-ops* and stores them in the trace cache, from where they are read, decoded, renamed and steered to any of the backends. Figure 2b shows the details of one of the backends. Each of them has its own integer and floating point register files and issue queues along with a memory order buffer coupled with a data TLB and a first-level data cache. This clustered backend organization has been previously shown to be effective at reducing the backend temperature [8].

Micro-ops are first handled by the dispatch logic, where the steering unit decides the destination cluster. Once the destination cluster is decided, the logical output register is mapped into a free register of that cluster and the instruction is dispatched.



(a)



(b)

Figure 2. (a) Block diagram of the clustered microarchitecture (b) Backend detail

After being dispatched, instructions remain in an issue queue until their inputs become available, and then, they are executed and results are written back to the register file. Special *copy* instructions are in charge of communicating register values between backends through a point-to-point link [6][23]. Each hop between tow neighbor clusters in the link requires 1 cycle.

Data caches are distributed and a load can be steered to any cluster. If there is a cache miss, the UL2 is accessed using the memory bus and the line is written in the cache of the cluster where the requesting load resides [13]. Store instructions are steered like any other instruction in order to compute their effective addresses, but a slot is allocated in all memory order buffers in order to disambiguate them from subsequent loads [2]. When the store address is computed, it is sent through the disambiguation bus and copied to all clusters, so disambiguation can be performed locally.

Integer and floating point instructions leave its issue queue after being issued. Store instructions remain in the memory order buffer until they commit and loads are kept in the memory order buffer until they are disambiguated. After being executed, instructions send a completion signal to the reorder buffer and they can be committed once they reach the head of the buffer.

2.1. Power Model and Temperature Models

This section briefly outlines the models for dynamic power, leakage power and the temperature that have been used to conduct the experiments.

Our dynamic power model is very similar to those existing in the literature [5]. Basically, an activity counter is associated to each functional block (e.g. register file, data cache, etc) and, in order to compute the energy, the activity counter is multiplied by its corresponding energy-per-operation value.

For each functional block of the processor, leakage power has been modeled as the average dynamic power multiplied by a factor dependent on the temperature. It is assumed that leakage power is going to be roughly 30% of dynamic power at ambient, inside box temperature (45º [19][27]). This percentage is increased as a function of temperature, in order to establish an exponential dependence between temperature and leakage.

The temperature model is similar to the one by Skadron *et al.* [26][27]. It is based on the duality of the thermal and the electrical phenomena. The temperature is estimated using an *RC* model that represents the heat transfer, also known as dynamic compact model (dynamic because it includes thermal capacitors modeling the timing response of the system).

Power and thermal models have been validated using both internal and public domain data. More details about the models can be found in [8].

3. Frontend Distribution

3.1. Distributed Renaming and Commit

In conventional microarchitectures, the rename table and the reorder buffer are monolithic structures. In order to sustain a high bandwidth both of them are complex multi-ported structures (especially for wide-issue processors). Such high activity translates into a high power density and, thus, high temperature. To make them more thermally efficient, we propose to distribute such structures into *N* different clusters. Each one of the frontend clusters (or frontends for short) feeds a subset of the backend clusters (or backends for short). For instance, assume a bi-clustered frontend, quad-clustered backend architecture (Figure 3): frontend 0 feeds backend 0 and 1 whereas frontend 1 feeds backend 2 and 3. In the proposed clustered frontend, instructions are fetched and decoded in the same way as in a centralized frontend. The steering engine (Figure 3-A) is kept centralized. It only requires a table with as many entries as number of logical registers. Each entry stores a bit per cluster , indicating whether there is a copy of that register in the cluster. Then, after the steering logic decides the destination backend, the instruction is directed to the frontend assigned to the chosen backend.

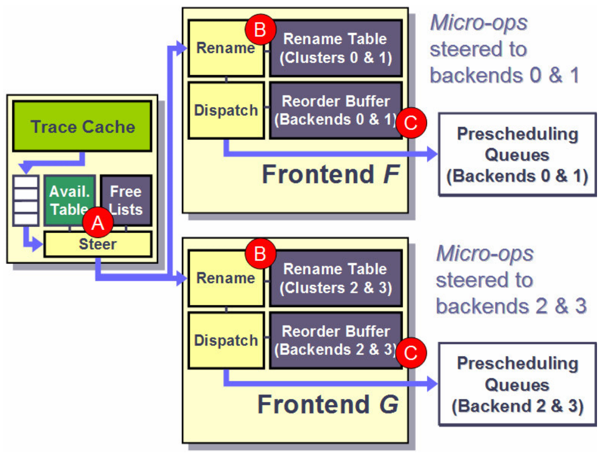

Trace Cache
Avail. Table
Free Lists
A
Steer
Rename
B
Rename Table (Clusters 0 & 1)
Dispatch
Reorder Buffer (Backends 0 & 1)
C
Frontend F
Micro-ops steered to backends 0 & 1
Prescheduling Queues (Backends 0 & 1)
Rename
B
Rename Table (Clusters 2 & 3)
Dispatch
Reorder Buffer (Backends 2 & 3)
C
Frontend G
Micro-ops steered to backends 2 & 3
Prescheduling Queues (Backend 2 & 3)

Figure 3. Rename table and reorder buffer distribution

Each one of the frontends has its own rename table and reorder buffer (Figure 3-B and Figure 3-C): the rename table stores the mappings only for its associated backends and the reorder buffer only holds the instructions that have been steered, but not yet retired, to its assigned backends. A disjoint rename table allows for a frontend clustering with no impact in the total latency, since there is no communication among clusters in order to rename registers. On the other hand, smaller rename tables and reorder buffers may have reduced access time. The delay of some wires might be increased but since each partition of the frontend is coupled to a subset of backends the communication costs from frontend to backend are reduced. Anyway, a blind physical distribution would increase the delays and the length of the dispatch stage if banks are far apart.

Two extensions to the current frontend are needed in order to complete the scheme: support for distributed renaming and distributed committing of instructions.

3.1.1. Distributed Renaming

The original centralized rename table (Figure 4) is split into several rename tables (Figure 5). In order to implement distributed register renaming with no communication among the different rename tables, some minor modifications are applied in the rename process. First, the renaming of the output register of any instruction is carried out at the steering stage (Figure 3-A), so that the *freelists* (one bit vector per backend) are kept centralized along with the steering logic. Right after the steering unit selects the destination backend, the corresponding *freelist* is accessed in order to obtain a free physical register to be assigned to the destination logical register.

Logical Register	Cluster 0	Cluster 1	Cluster 2	Cluster 3
Ri	Pj,0	Pk,1	Pl,2	Pm,3
…				

Figure 4. Centralized rename table

Second, when a *copy* instruction is needed, two cases can be distinguished. Suppose a given instruction is directed to frontend *F*. If the value needed is present in any of the backends assigned to *F*, the *copy* is processed normally. When the copy has to be sent to a backend belonging to some other frontend, a *copy request* mechanism is needed. *Copy request* is a two steps process:

1. The request is generated (Figure 3-A):
   - A destination register for the value is obtained from the corresponding *freelist.*
   - The request is sent to frontend *G* (where at least one of its backends holds the value). It indicates the logical register to be copied as well as the destination physical register and backend.
2. Frontend *G* generates the copy (Figure 3-B).

The hardware required to check whether a register is mapped into a cluster is very simple: an "availability table" that has as many entries as number of logical registers and as many bits per entry as number of backends. Each one of these bits indicates that the logical register has a valid copy in that backend. Note that this is not the actual rename table. Still in the steering stage, mappings for destination registers are written in the rename table.

The actual mapping of source registers into physical registers is done independently in each one of the frontends (Figure 3-B).

The simplicity of this mechanism is such that no impact is expected on the frontend latency. The only overhead of this mechanism is related with the *copy request* (a signal between frontends) generation. However, this task can be done in parallel with the destination register renaming. Moreover, latency can be hidden because the smaller structures resulting from the distribution mechanism are expected to have smaller latency.

Logical Register	Cluster 0	Cluster 1
Ri	Pj,0	Pk,1
…		

Logical Register	Cluster 2	Cluster 3
Ri	Pl,2	Pm,3
…		

Figure 5. Distributed rename table

3.1.2. Distributed Commit

Since the reorder buffer is distributed, a mechanism is needed in order to synchronize instructions that are being committed (Figure 3-C). Each frontend owns a portion of the reorder buffer that contains the instructions that were steered to any of the backends of that frontend. A simple modification in the reorder buffer structure is needed in order to support the distribution.

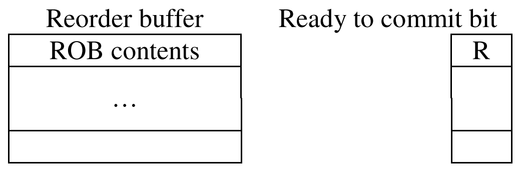


Figure 6. Centralized reorder buffer

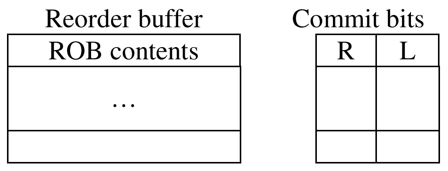


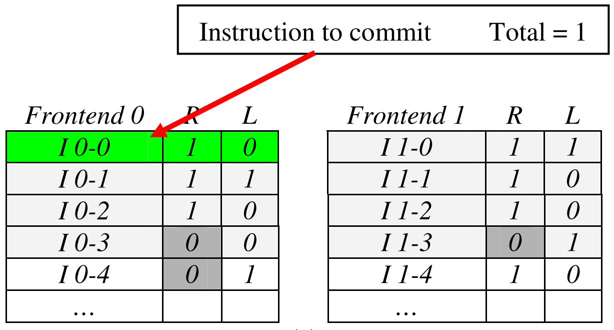
Figure 7. Distributed reorder buffer

In a traditional reorder buffer (Figure 6) a bit indicates whether an instruction is ready (*R*) to commit or not. Committing implies reading from the reorder buffer some of the oldest entries in order to free the physical registers that are no longer needed.

In the distributed version each frontend has a partial reorder buffer (Figure 7). A local extra field *L* indicates the reorder buffer where the next instruction in program order is located (it can be the same or any other reorder buffer). A special register points to the reorder buffer that holds the next instruction to be committed. By examining the *R* and *L* bits a simple logic can decide how many instructions must be committed from each frontend. First, the *C* older *R-L* fields of each reorder buffer are read, where *C* is the commit bandwidth. Then, the *R-L* pairs of the reorder buffer that holds the oldest instruction are checked:

- If *R*=0 then no more instructions are committed.
- If *R*=1 and *L* points to the current reorder buffer, the instruction is selected to be committed and the next instruction in the same reorder buffer is analyzed.
- If *R*=1 and *L* points to another reorder buffer the instruction is selected to be committed and the next instruction to be analyzed is selected from the reorder buffer pointed by *L*.
- This process is repeated until *C* instructions have been selected to be committed.

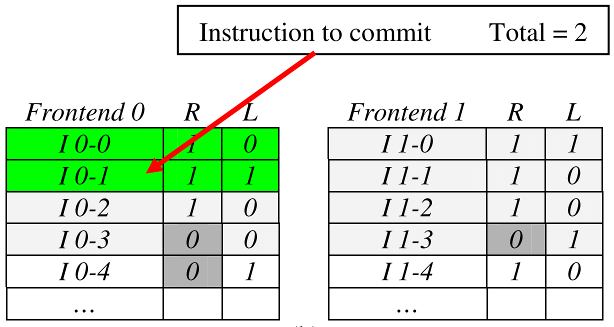
Once the instructions to be committed are identified, the corresponding entries of the various reorder buffers are read.



Frontend 0	R	L
🟩 *I 0-0*	*1*	*0*
I 0-1	*1*	*1*
I 0-2	*1*	*0*
I 0-3	*0*	*0*
I 0-4	*0*	*1*
…		

Frontend 1	R	L
I 1-0	*1*	*1*
I 1-1	*1*	*0*
I 1-2	*1*	*0*
I 1-3	*0*	*1*
I 1-4	*1*	*0*
…		

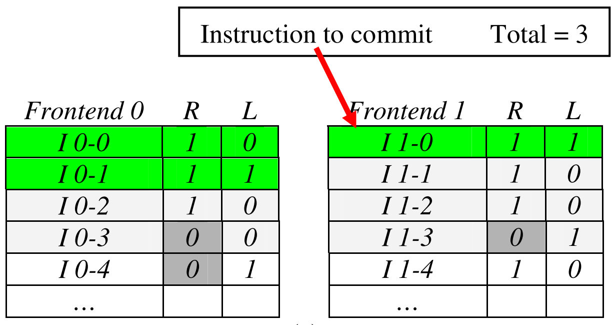
(a)



Frontend 0	R	L
🟩 *I 0-0*	*1*	*0*
🟩 *I 0-1*	*1*	*1*
I 0-2	*1*	*0*
I 0-3	*0*	*0*
I 0-4	*0*	*1*
…		

Frontend 1	R	L
I 1-0	*1*	*1*
I 1-1	*1*	*0*
I 1-2	*1*	*0*
I 1-3	*0*	*1*
I 1-4	*1*	*0*
…		

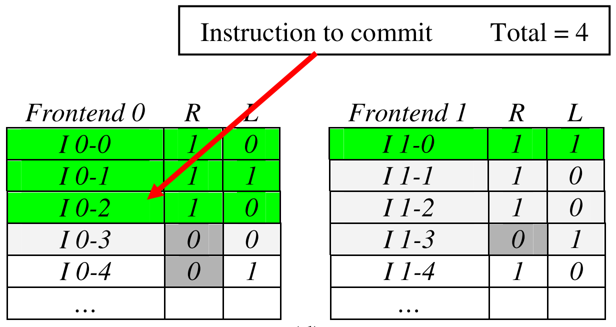
(b)



Frontend 0	R	L
🟩 *I 0-0*	*1*	*0*
🟩 *I 0-1*	*1*	*1*
I 0-2	*1*	*0*
I 0-3	*0*	*0*
I 0-4	*0*	*1*
…		

Frontend 1	R	L
🟩 *I 1-0*	*1*	*1*
I 1-1	*1*	*0*
I 1-2	*1*	*0*
I 1-3	*0*	*1*
I 1-4	*1*	*0*
…		

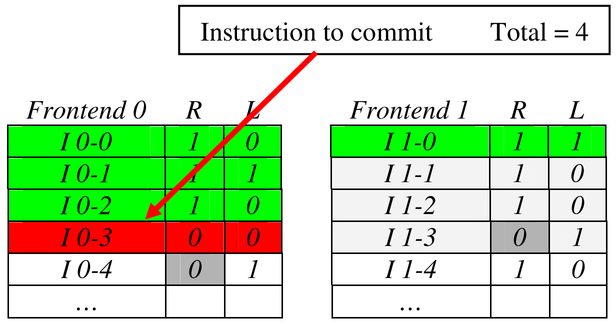
(c)



Frontend 0	R	L
🟩 *I 0-0*	*1*	*0*
🟩 *I 0-1*	*1*	*1*
🟩 *I 0-2*	*1*	*0*
I 0-3	*0*	*0*
I 0-4	*0*	*1*
…		

Frontend 1	R	L
🟩 *I 1-0*	*1*	*1*
I 1-1	*1*	*0*
I 1-2	*1*	*0*
I 1-3	*0*	*1*
I 1-4	*1*	*0*
…		

(d)



Frontend 0	R	L
🟩 *I 0-0*	*1*	*0*
🟩 *I 0-1*	*1*	*1*
🟩 *I 0-2*	*1*	*0*
🟥 *I 0-3*	*0*	*0*
I 0-4	*0*	*1*
…		

Frontend 1	R	L
🟩 *I 1-0*	*1*	*1*
I 1-1	*1*	*0*
I 1-2	*1*	*0*
I 1-3	*0*	*1*
I 1-4	*1*	*0*
…		

(e)

Figure 8. Example of distributed committing

Figure 8 shows an example for a commit bandwidth of 4 instructions per cycle. Instructions are processed following the previous algorithm until a “not-ready to commit” one is found (i.e. *I 0-3*) or until the commit bandwidth is reached (four instructions). It is assumed that *L*=0 means that the next instruction is in the same reorder buffer (*L*=1 otherwise).

Since the commit complexity has been increased, its delay might increase. Therefore, in the simulation section, the commit latency will be increased by 1 cycle.

3.2. Sub-banked Thermal-Aware Trace Cache

An important part of the front-end area is devoted to the instruction cache (in our case a trace cache). Two orthogonal mechanisms are evaluated in order to reduce peak temperatures in the trace cache.

The trace cache is supposed to be divided into banks with non-overlapping contents. This allows extra flexibility to relocate some blocks in the frontend's layout in order to surround hot blocks of cold blocks.

3.2.1. Bank Hopping

The term bank hopping refers to V_{dd}-gating one or more of the cache banks during a given interval of time in a rotating manner. It is based on the idea of migrating activity to reduce average power density over time [15]. Contents of a V_{dd}-gated bank are lost, so, when reconfiguring, the mapping function must be changed to steer all accesses previously mapped into that bank to a new enabled bank.

In order not to reduce the total effective cache size, we add an extra bank to the ones already present in the trace cache to do the hopping. This way the total area devoted to the trace cache is increased, but the power is not (except for the misses caused because of hopping), since always one of the banks is gated. We have evaluated the effect of devoting this extra area to hold traces, observing minor performance improvements.

3.2.2. Thermal-Effective Bank Mapping Function

Banking is a well known technique, applied to caches in general, in order to reduce complexity, power dissipation and area. These cache organizations attempt to balance accesses across banks in order not to increase the miss rate. However, we have observed that, from a temperature standpoint, balancing the accesses in a banked trace cache entails an important temperature imbalance across banks due to floorplan issues and access bursts (i.e. a balanced mapping function can distribute accesses across banks effectively in the long term but may be inefficient in the short term, stressing one of the banks). Therefore, any of the banks may become a hot spot or there may be a high temperature imbalance.

To solve these thermal inefficiencies, we propose a new bank mapping function with the purpose of distributing the accesses not in a balanced way but in a thermally effective way.

Whenever the trace cache is accessed, a mapping function selects the bank where the line is going to be inserted. When sub-banking, the selection policy performs a bitwise XOR of two five-bit fields of the trace cache address (composed of the branch bits plus the *PC* of the first instruction of the trace) to obtain a five bit number. The set of bits was picked to obtain the best distribution of addresses over bit combinations.

These five bits are used to index a table that holds in each entry the bank assigned to that particular combination. For *N* banks, a "balanced" distribution would assign *1/N* of the combinations to each bank. That is, entries from 0 to 15 would point to bank 0 whereas entries from 16 to 31 would point to bank 1. The thermal-aware mapping function modifies the distribution of entries among banks, assigning more entries of the table to the colder banks, so at least a thermal sensor is required on each bank. This way a biased mapping function is implemented. At every given interval (10M cycles in our experiments) the mapping table is recalculated depending on current temperatures of the different banks (Figure 9). The number of entries that best match the requirements (because of temperature) are allocated to each bank.

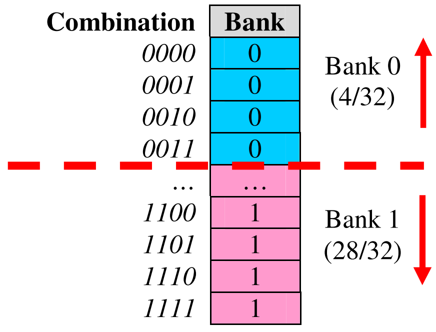


Figure 9. Bank mapping table for two banks

Experimentally it was found that the activity of a bank should be divided by a factor of two, for each 3ºC of difference between the temperature of a bank and the average temperature of all banks.

4. Evaluation

Experiments have been conducted using an execution-driven simulator that runs IA32 binaries. The processor can fetch, dispatch and commit up to 8 *micro-ops* per cycle. Table 1 summarizes the main parameters of the baseline monolithic architecture.

We have run 26 SPEC2000 applications for the evaluation process. Each execution trace (from the *test* input set) is divided in ten equal-size slices (i.e., slices of different applications have different size) and the fourth of them is selected to be executed in the simulator (the whole slice or up to 200 million instructions). When this slice division ends up with traces shorter than 200M instructions (which happens for *eon, gap, gcc, gzip, perlbmk, vortex, swim, fma3d,* and *ammp*) the program is divided into slices of 200M instruction. At the end all traces were 200M instructions long except *eon, fma3d, mcf, perlbmk* and *swim*, whose whole trace was shorter, which were run for 127, 30, 156, 58 and 112 millions of instructions respectively.

As far as the thermal model is concerned, at the beginning of the simulation we assume that the processor has already been running for a long time dissipating its nominal average dynamic power (obtained for 50M instructions), and the leakage corresponding to its temperature, until temperature converges or reaches the emergency limit (381K). In this way, simulations are started with the processor already warm. Then, during normal execution, every ten million cycles the temperature is updated using the per-block dissipated power. When using the thermal-aware mapping function, the power of the trace cache is supposed to be the proportional part of the total cache power (as if all banks had the same activity) in order not to benefit any bank.

Table 1. Processor configuration

Frontend	
Trace cache/Fetch	32K *micro-ops*, 4-way, 4 cycle fetch-to-dispatch latency
Decode, rename and steer	8 cycles (regardless of the destination cluster)
UL2	2 MB/8-way, 12 cycle hit, 500+ miss
Communi-cations	2 memory buses, 2 disambiguation buses, 4-cycle latency + 1-cycle arbiter, 2 bidirectional p2p link (1 cycle per hop; 2 from side to side of the chip)
Each backend	
Queues	40-entry IQueue 1 inst/cycle, 40-entry FPQueue 1 inst/cycle, 40-entry CopyQueue 1inst/cycle , 96-entry MemQueue 1inst/cycle, 10 cycle dispatch latency; 20 entries per prescheduler queue
Register file	160 int. registers (6 read and 3 write ports) and 160 FP registers (5 read and 3 write ports)
Data cache	16 KB/2-way, 1 cycle hit, 1 read port, 1 write port, write update

Figure 10 shows the floorplan of the processor. We assume a processor designed at 65nm, running at 10GHz, with a V_{dd} of 1.1V. Areas were computed using an enhanced version of Cacti [25] for cache-like structures, and scaling down the rest of the structures from current designs. The thermal solution attached to the die of the processor consists of a copper heat spreader, in contact with the die, whose size is 3.1x3.1x0.23cm (similar to the one used in Pentium® 4 Northwood processors [17]). On top of it there is a copper heat sink of 7x8.3x4.11cm [17].

In all figures, the baseline is a quad-cluster processor with unified renaming and commit, and a two-banked trace cache with no thermal-aware bank selection policy. Temperature improvements are measured as the reduction on the temperature increase over ambient (45ºC). The metrics presented are:

- *AbsMax*: Peak temperature.
- *Average*: Average temperature over time and space.
- *AvgMax*: Average of the maximum temperatures obtained in each interval.

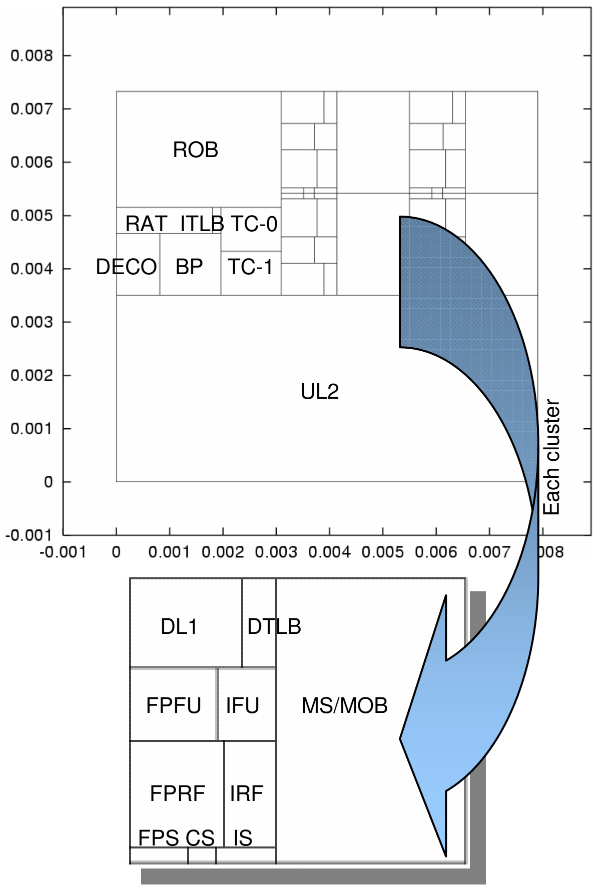


Figure 10. Floorplan of 2-banked trace cache baseline processor (a) Processor (b) Cluster details

The frontend floorplan for bank hopping is shown in Figure 11. In all the cases the floorplans are intended to keep constant the aspect ratio (because that affects the ability to spread heat laterally) of the critical blocks (in our case, the schedulers). Floorplans for distributed commit are not shown due to lack of space, but both ROB and RAT partitions are kept together in the same location as in the original centralized version.

We have not enabled any mechanism to be is triggered at a thermal emergency (it is part of our future work). As stated before, whenever a peak temperature is reached, a technique in charge of cooling down the processor is triggered. Such mechanisms degrade performance and techniques reducing peak temperatures would reduce the number of times that these mechanisms are initiated. Including such mechanisms would improve our results, since any technique that reduces the peak temperature may experience smaller slowdowns and even speedups.

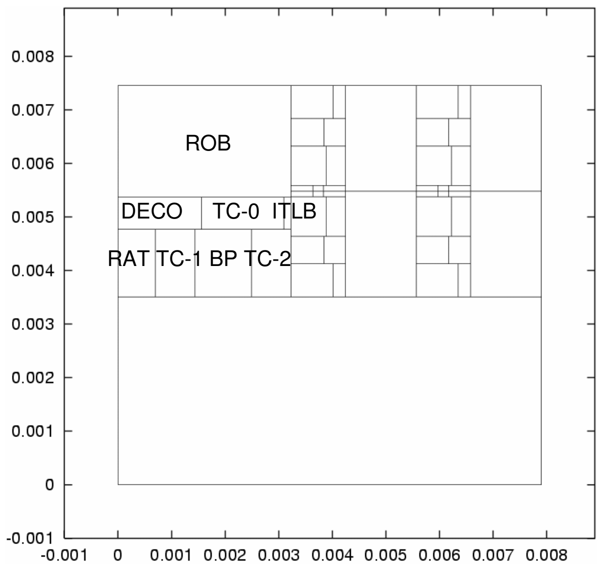


Figure 11. Floorplan for two-banked trace cache for bank hopping configurations

4.1. Distributed Renaming and Commit

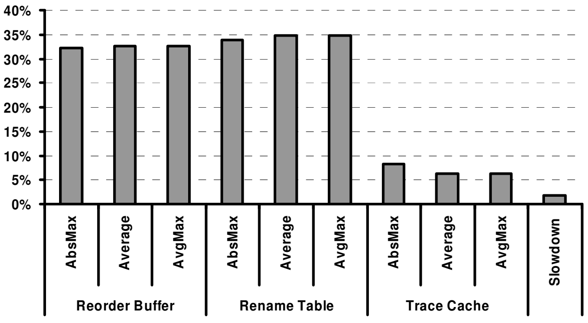


Figure 12. Reduction of temperature for the distributed renaming and commit

Figure 12 shows the results for the distributed renaming and commit technique. The results show the average for the 26 SPEC applications (all of them follow the same trend).

It can be seen that the technique drastically reduces the temperature of both the reorder buffer and the rename table with a slowdown of only 2%. It indirectly reduces the trace cache temperature because of heat spreading. The area overhead is 3% over the total processor area, and the temperature reductions are 32% and 34% for the peak temperature of the reorder buffer and rename table, respectively, and 33% and 35% for the average. The benefit does not come from the area increase (i.e. inserting a piece of bulk silicon of that size does not obtain such reductions), but from the reduction in the power density due to the distribution of the activity and the reduction in the energy per access. For instance, the distributed ROB reduces power by 11% on average. This is due to the reduction in its complexity (each access consumes less than half the energy that consumed in the centralized version).

4.2. Thermal-Aware Trace Cache

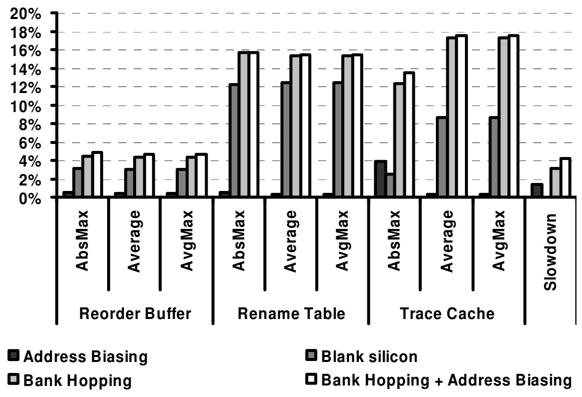


Figure 13. Sub-banked trace cache temperature improvements

In Figure 13 the results for trace cache techniques are depicted. The biased mapping function alone reduces the peak temperature of the trace cache (by 4%) since the activity is effectively spread across the banks as a function of the temperature. However, the average temperature is not reduced since the activity is spread but not reduced. The slowdown is only 2%.

The benefit on the trace cache increases when bank hopping is considered. The trace cache average temperature is decreased by 17% and the peak temperature by 12%. This allows the rename table to dissipate part of its heat towards the trace cache, achieving a reduction in the peak and average temperature of 16% and 15% respectively. The slowdown due to bank invalidation when hopping is only 3% (the hit ratio is reduced less than 1%). Area overhead is 1.6% over total processor area.

For comparison purposes, a configuration including blank silicon (1 out of the 3 banks statically gated) is included. It can be seen that the proposed techniques outperform this option.

The combination of bank hopping and a thermal-aware mapping function achieves temperature reductions of 14% for the peak temperature and 18% for the average temperature with a slowdown of only 4%.

4.3. Distributed Frontend

Figure 14 shows the results when the distributed rename and commit is combined with a thermal-aware trace cache. Results for each individual technique are also presented for comparison purposes.

Combining both techniques achieves a synergistic effect. It is interesting to observe that reducing the temperature of some area, affects positively the blocks placed around it. For instance, distributing the reorder buffer and decreasing its temperature also decreases trace cache temperature since part of its heat is spread towards the reorder buffer. On the other hand, the opposite effect may also appear. For example, when trace cache hopping is applied along with distributed rename and commit, the temperature reductions of the rename table are lower because the trace cache is actually spreading heat towards it (as the thermal maps show). This did not happen (or at least, not as much) in the configuration without bank hopping.

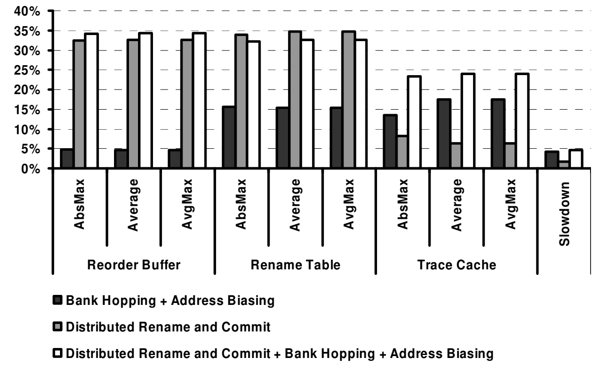


Figure 14. Overall temperature results for the distributed frontend

Nevertheless, the combination of distributed rename and commit along with a thermally effective trace cache is a very effective approach to reduce the frontend temperature. The temperature of the reorder buffer, rename table and trace cache are reduced by 35%, 32% and 25% respectively.

5. Related Work

Controlling temperature through microarchitectural techniques is a fairly new area. Huang *et al.* [16] propose a framework to maximize energy savings and to guarantee that temperature remains under a certain threshold. The framework combines a number of energy-management techniques, such as voltage-frequency scaling, and sub-banking of the data cache, among others. Brooks and Martonosi [5] propose a set of control techniques evaluated on top of different triggering mechanisms, with the aim of reducing thermal emergencies. They use the average power in an interval as a proxy for temperature. Skadron *et al.* [26][27] propose a thermal simulator based on the duality between heat transfer and the electrical phenomena. Several techniques are proposed to control peak temperature and to reduce thermal emergencies: PID controllers, frequency scaling, fetch toggling, and register file replication. Lim *et al.* [18] propose a secondary ultra-low power pipeline that is used when a given temperature threshold is exceeded. Asanović *et al.* [15] study the impact of activity migration among replicated units on power density. Donald *et al.* address design issues for SMT and CMP architectures [10], and Ghiasi *et al.* for dual-core processors [12]. Some current commercial processors such as the Pentium® M implement thermal monitors to control the temperature of the chip [24].

6. Conclusions

Keeping silicon at an operating temperature is becoming more challenging and expensive as the power density of microprocessors keeps increasing. Higher temperatures increase the cost of the package and the thermal solution of a processor, increase its leakage power, and penalize its performance. This paper addresses the issue of temperature in the frontend of a clustered microarchitecture, which is an important contributor of the total heat dissipated by the processor.

A thermally efficient frontend is proposed and analyzed. First, a mechanism to distribute the rename and commit logic is shown to reduce temperature by more than 30% (both peak and average temperatures) in the rename table and in the reorder buffer, with a small impact on performance (only 2%). In order to reduce temperature in the trace cache a banked design with a bank hoping scheme is proposed. The trace cache is enhanced with a thermal-aware mapping that attempts to balance temperature among cache banks. Experiments show reductions of 14% for the maximum and 17% for the average temperature. When both techniques are combined together, the temperature of the reorder buffer, rename table, and trace cache is reduced by 35%, 32% and 25% respectively.

Acknowledges

The authors would like to thank Kevin Skadron, for its comments in the thermal model, Aviad Cohen for the help on the validation and Ronny Ronen for his comments on the draft version of the paper. This work has been partially supported by the *Ministerio de Educación y Ciencia* and FEDER funds, under grant TIN2004-03072.

References

[1] V. Agarwal, M.S. Hrishikesh, S.W. Keckler and D. Burger. “Clock Rate versus IPC: the End of the Road for Conventional Microarchitectures“. In Proceedings of the 27th International Symposium on Computer Architecture, 2000.

[2] R. Balasubramonian, S. Dwarkadas and D.H. Albonesi. “Dynamically Managing the Communication Parallelism Trade-off in Future Clustered Processors. “. Proceedings of the International Symposium on Computer Architecture, 2003.

[3] M. Bohr. “Interconnect Scaling - the Real Limiter to High-Performance ULSI”. In Proceedings of the International Electron Devices Meeting, pp. 241-244, Dec. 1995.

[4] S. Borkar. “Design Challenges of Technology Scaling”. IEEE Micro, 19(4), pp. 23-29, 1999.

[5] D. Brooks, V. Tiwari V. and M. Martonosi. “Wattch: A Framework for Architectural-Level Power Analysis and Optimizations”, in Proceedings of the 27th International Symposium on Computer Architecture, pp. 83-94, 2000.

[6] R. Canal, J.M. Parcerisa and A. González. “A Cost-Effective Clustered Architecture” Proceedings of the International Conference on Parallel Architectures and Compilation Techniques, 1999.

[7] P. Chaparro, J. González and A. González. “Thermal-Aware Clustered Microarchitectures”. ICCD 2004.

[8] P. Chaparro, J. González and A. González. “Thermal-Effective Clustered Microarchitectures”. TACS Workshop at ISCA-31, June 2004.

[9] V. De and S. Borkar. “Technology and Design Challenges for Low Power and High Performance”. Proceedings of the International Symposium on Low Power Electronics Design pp. 163-168, 2000.

[10] J. Donald and M. Martinosi. “Temperature-Aware Design Issues for SMT and CMP Architectures” WCED Workshop at ISCA-31, June 2004.

[11] K. Farkas, P. Chow, N. Jouppi and Z. Vranesic. “The Multicluster Architecture: Reducing Cycle Time through Partitioning”. Proceedings of the International Symposium on Microarchitecture, 2000.

[12] S. Ghiasi and D. Grunwald. “Design Choices for Thermal Control in Dual-Core Processors”. WCED Workshop at ISCA-31, June 2004.

[13] J. González, F. Latorre and A. González. “Cache Organizations for Clustered Microarchitectures”. In Proceedings of the third Workshop on Memory Performance Issues at ISCA 04.

[14] S. Gunther, F. Binns, D. M. Carmean and J.C. Hall. “Managing the Impact of Increasing Microprocessor Power Consumption”. Intel Technology Journal, Q1, 2001.

[15] S. Heo, K. Barr, K. Asanović “Reducing power density through activity migration” Proceedings of the 2003 International Symposium on Low Power Electronics and Design, 2003.

[16] M. Huang, J. Renau, S-M. Yoo and J. Torrellas. “A Framework for Dynamic Energy Efficiency and Temperature Management”. Proceedings of the International Symposium on Microarchitecture, pp. 202-213, 2000

[17] Intel Corporation “Intel® Pentium ® 4 Processor in the 423-pin Package Thermal Solution Functional Specification” http://www.intel.com/design/pentium4/guides/249204.htm.

[18] C. H. Lim, W. R. Daasch, G. Cai, “A Thermal-Aware Superscalar Microprocessor” Quality Electronic Design, 2002. Proceedings. International Symposium on , 18-21 March 2002.

[19] R. Majan “Thermal management of CPUs: A perspective on trends, needs and opportunities”, Oct. 2002. Keynote presentation, THERMINIC-8.

[20] S. Manne, A. Klauser and D. Grunwald. “Pipeline Gating: Speculation Control For Energy Reduction”. In Proceedings of the 25th Annual International Symposium on Computer Architecture, June 1998.

[21] D. Matzke. “Will Physical Scalability Sabotage Performance Gains?” Computer Magazine, Vol. 30, No. 9, pp 37-39.

[22] J. Moore, R. Sharma, R. Shih, J. Chase, Ch. Patel and P. Ranganathan. “Going beyond CPUs: The Potential of Temperature-Aware Solutions for the Data Center”. In Proceedings of the First Workshop of Temperature-Aware Computer Systems (TACS-1) held at ISCA 04.

[23] J.-M. Parcerisa, J. Sahuquillo, A. González, J. Duato "Efficient Interconnects for Clustered Microarchitectures" Proc. of the Int. Conf. on Parallel Architectures and Compilation Techniques PACT 2002.

[24] E. Rotem, A. Naveh, M. Moffie and A. Mendelson. “Analysis of Thermal Monitor Features of the Intel Pentium M Processor”, TACS Workshop at ISCA-31, June 2004.

[25] P. Shivakumar, N. P. Jouppi “CACTI 3.0: An Integrated Cache Timing, Power and Area Model” WRL Research Report 2001/2.

[26] K. Skadron, T. Abdelzaher and M. Stan. “Control-Theoretic Techniques and Thermal-RC Modelling for Accurate and Localized Dynamic Thermal Management”. Proceedings of the International Symposium on High Performance Computing, 2002.

[27] K. Skadron, M. R. Stan, W. Huang, S. Velusamy, K. Sankaranarayanan, and D. Tarjan. “Temperature-Aware Microarchitecture”. In Proceedings of the 30th Annual International Symposium on Computer Architecture, April 2003.

[28] L.-T. Yeh and R.C. Chu. “Thermal Management of Microelectronic Equipment: Heat Transfer Theory, Analysis Methods and Design Practices”. ASME Press, New York, NY, 2002.

[29] H. de Vries “Looking at Intel's Prescott die”. http://chip-architect.com/news/2003_03_06_Looking_at_Intels_Prescott.html

[30] Y. Zhang, D. Parikh, K. Sankaranarayanan, K. Skadron and M. Stan. “Hotleakage: A Temperature-Aware Model of Subthreshold and Gate Leakage for Architects”. Technical Report CS-2003-05, University of Virginia Department of Computer Science, Mar. 2003. 34