Thermal-Aware Instruction Assignment for VLIW Processors[1]

Benjamin Carrión Schäfer and Taewhan Kim
Department of Electrical Engineering and Computer Science
Seoul National University, Seoul, Korea
{schaferb, tkim}@ssl.snu.ac.kr

***Abstract*— Very Long Instruction Word Processors (VLIW) allow to execute multiple instructions in parallel. At the same time VLIW compilers optimize the source code for maximum performance by grouping as many instructions in parallel as possible. This work addresses the problem of thermal-awareness in VLIW compilers with a temperature control and reduction techniques with an objective of minimizing the peak temperature in the VLIW processor's functional units. We present a main approach as well as an improved fast version and compare them with a technique that inserts NOPs in the assembly code to allow the hottest units to cool down. The main technique, called temperature-aware instruction binding technique TempIB, effectively binds the instructions executed in parallel to the coolest possible functional units for a given fixed schedule. It generates, for each instruction in a scheduled instruction word, a priority queue of the coolest functional units that can execute the instruction, and rebinds it to the coolest possible unit, considering the temperature as well as the power consumed by the instruction. Then, an improved version called TempIB-f, which exploits the power density and local temperature of each unit to minimize the number of thermal simulations needed, accelerating the run time dramatically, is proposed. From experimentation using a set of benchmark designs, it is confirmed that our temperature reduction techniques are effective, lowering down the peak temperature of the initial design by up to 13.82% and 12.79% by TempIB and TempIB-f, respectively.**

I. Introduction

Very Long Instruction Word (VLIW) is a concept for processing technology that dates back to the early 1980s. The term VLIW refers to the size of each instruction that is carried out by a processor. This process allows multiple operations to be executed simultaneously to achieve a maximum utilization of processing power. The VLIW code is ordered for the processor at compile time. The hardware generally consists of identical multiple execution units. VLIW was nearly un-implementable at its early stages due to the prohibitively expensive nature of memory at the time. The architecture is ideal for quick computation of complex and repetitive algorithms. VLIW's advantages come largely from having an intelligent compiler that can schedule many instructions simultaneously maximizing the total instruction level parallelism. In our work we incorporate temperature awareness to the compiler avoiding the appearance of hotspots as well as trying to flatten the processor's functional units (FUs) temperature distribution.

With the advent of new technology and scaling design parameters, total performance is not the only parameter that needs to be addressed when creating a new design. Power and now temperature are becoming increasingly as important factors as performance, especially in embedded systems. In this paper we address the temperature issue in VLIW processors. Property vendor compilers do only consider maximum performance when compiling the source code, but do not address the temperature issue.

Temperature has an adverse effect on multiple aspects. It affects the lifetime of the integrated circuit by accelerating the chemical process taking place inside of the chip following Arrhenius equation. Studies show the mean time between failure (MTBF) of an IC is multiplied by a factor 10 for every 30^0C rise in the junction temperature [1]. Secondly leakage power is becoming the dominant source of power consumption for new process technologies [2] which grows exponentially with temperature. Moreover, temperature has a negative effect on carrier mobility and therefore switching speed of the transistors and thus the overall timing of the circuit. Consequently it is highly desirable to have an even temperature distribution on the chip in order to avoid costly re-design due to timing/temperature as well as simplifying the verification phase. Furthermore, expensive heat dissipaters are required to maintain the chip at a reasonable temperature and in case of embedded systems cannot be possibly used. Studies have reported that above 30-40 Watts (W), additional power dissipation increases the total cost per chip by more than $1/W [3].

Temperature is highly dependent on power consumption but depends on a multiple of other factors, making power alone not a valid measure for temperature. Temperature also depends on the placement of the units in the chip. Placing heavy power consuming units close together will intuitively generate an even higher temperature area in the chip as temperature is additive in nature. In contrast, placing power consuming units close to units that have a moderate power consumption will allow the heat generated to dissipate through these units. Other aspects that affect temperature are the execution order of tasks in a unit. Executing tasks one after the other will help the temperature build up whereas spacing the execution of tasks in a unit will allow the unit to have a time to prevent it from heating up. Consequently, temperature should be addressed as an individual design parameter.

In the case of microprocessors as well as DSPs one could envision a system where tasks are assigned dynamically based on the temperature of each functional unit. Each functional unit, or units where a hotspot is more likely to happen would have a thermal sensor. [4] proposed a technique to minimize the number of these thermal sensors. The reading of these

[1] This research work has been supported by Nano IP/SoC Promotion Group of Seoul R&BD Program in 2007. This work was also partially supported by the Ministry of Science and Technology (MOST)/Korea Sceince and Engineering Foundation (KOSEF) through the Advanced Information Technology Research Center (AITrc).

sensors could then be used to determine which units should be selected to execute the instruction and which units should be turned off to allow these to cool down. Although the area used by these sensors is small they complicate the power supply grid design. An excessive number of thermal sensors would cause the power supply to interfere significantly with the power grid design for the rest of the processor. (e.g. the *Cell Processor* requires separate power supply grid for the thermal sensors to achieve the required accuracies [5].) It is therefore more than desirable to have static techniques that do not yield to any hardware overhead. The contributions of this paper are the following:

- Studies of the effects of temperature on VLIW architectures (in particular TI's C6X DSPs) and the introduction of temperature-awareness at the compiler.
- Introduction of a tightly integrated framework to study the thermal behavior of DSP architectures.
- Introduction of a new temperature reduction technique, (TempIB), and an improved fast version (TempIB-f), which consider the on-chip thermal distribution when assigning instructions to the different functional units (FUs), without any performance degradation.
- Comprehensive experimental results to validate the proposed techniques, compared to the initial thermal unaware compilation as well as to a technique that inserts NOPs in the assembly code to allow FUs to cool down.

This work is made on the assumption that heat flows laterally inside the chip, as shown in multiple previous works, especially thermal-aware floorplanners [6], [7], [8]. The influence of the lateral heat flow will depend on the thermal conductivity of the primary and secondary heat flows of the chip (heat flow through the package and through the pins respectively). In this case we are targeting mostly embedded systems where DSPs are normally used, which have very strict space constraints limiting the type and size of heatsink. In these cases lateral heat flow becomes extremely important. On the other hand being able to control the lateral heat flow allows also to use a cheaper package (with lower conductivity).

II. Motivational Examples

This section shows two motivational examples that clarify the need to incorporate temperature-awareness into the compilation flow of VLIW processors. A brief description of the target DSP architecture used in this paper is presented next, to help to understand the motivational examples.

Texas Instruments C6X family has eight FUs, and the data path can be divided into two groups of four [9]; Each FU in one data path is almost identical to the corresponding FU in the other data path. Table I shows the different FUs and the operations they can perform. A floorplan of these FUs can be seen in Fig. 1(a).

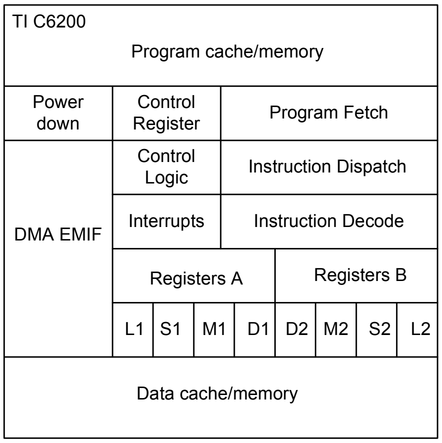

Fig. 1. An example of TI's C6X DSP floorplan.

TABLE I
Functional unites and instr. performed for TI's C6X family.

Functional Units	Operations
.L unit (.L1, .L2)	32/40-bit arithmetic and compare operations; 32-bit logical operations

.S unit (.S1, .S2)	32-bit arithmetic operations;
	32/40-bit shifts and 32-bit bit-field operations;
	32-bit logical operations; Branches,
	constant generation and register transfers

.M unit (.M1, .M2)	16×16-bit multiply operations
.D unit (.D1, .D2)	32-bit add, subtract and address calculation; Loads and stores with 5-bit constant offset; Loads and stores with 15-bit constant offset

Observation 1: Fig. 2 shows a snapshot of a thermal simulation of one of the benchmarks (IMG_CONV3X3). The temperature as well as the accumulative instructions executed on each of 4 FUs (out of total 8 to make the graph more readable) are plotted on the same graph with respect to time, showing the difference in temperature between two equivalent units as well as the load balance of each FU, for the initial performance-optimized compilation. It can be seen that there is a room for load balancing as well as for temperature reduction, by re-assigning instructions to other FUs.

Observation 2: Fig. 3 shows three instructions to be executed in parallel and the FUs in which they can be executed. Fig. 3(a) shows a result by random assignment of the instructions to the FUs. On the other hand, Fig. 3(b) show a result by thermal-aware reassignment of the instructions to the coolest available FUs. The instruction executed on the hottest FU (instr1) is selected first and bound to the coolest possible FU, which

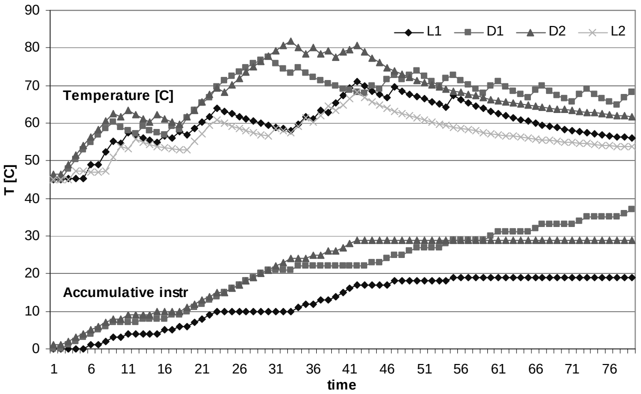

Fig. 2. Graph indicating the accumulative instructions of 4 different FUs of the TI C6X DSP and their temperature vs time for IMG_CONV3X3.

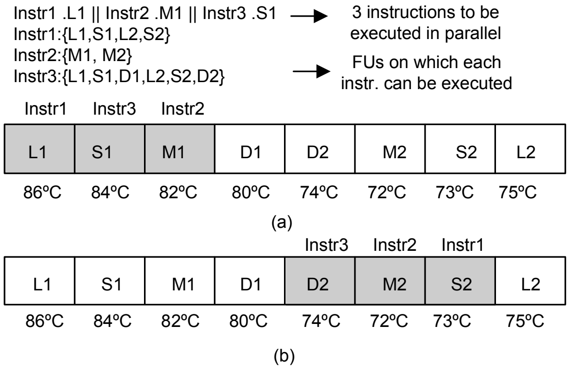

Fig. 3. Example showing the effect of instruction binding on the temperatures: (a) Result by a temperature-unaware binding; (b) Result by a temperature-aware rebinding.

is S2 in the example. The instruction assigned to the second hottest FU (instr3) is bound to the next coolest FU, which in this case is D2. The last instruction (instr2) is then moved to the coolest available FU by which it can be executed. Some restrictions need to be taken into account like the fact that some instructions can only be mapped to a single FU or set of FUs, in which case these have to be bound first. The register bank that needs to be accessed should also be taken into account as each FU reads directly from and writes directly to the register file within its own data path. On the C62x DSP, six of the eight FUs have access to the register file on the opposite side via a cross path.

III. Related Work

There has been some previous work targeting the reduction of overall temperature at the architectural level. A unified framework was presented in [11] in order to maximize energy savings and guarantee the temperature below a given threshold and performance penalty. Some other techniques tried to replicate some FUs and swap them with FUs used when the current FUs reached a certain maximum temperature, which was called *migration computing* [12]. The main problem of this approach is that once the threshold temperature is reached at a FU, the data items in the register file which is connected to the FU need to be copied every time the FU is migrated, resulting in performance degradation. In addition, the swapping between FUs continues with no guarantee that the timing constraint is met. Moreover, the replication of FUs increases the area. Another example of migrating computing is the dual-pipeline scheme proposed in [13] where a secondary scalar pipeline is added for energy efficiency. However, as indicated in [14] it incurs a large slowdown of performance. The work in [18] proposed a dynamic thermal management scheme which also satisfies a given worst-case power consumption on processors.

Multiple work has been performed at the compiler level with power-aware scheduling for VLIW [22], [23], [24], in which the slack in each operation is exploited to reduce the leakage power by putting unused units in a low power mode. However, theses approaches do not consider thermal information. Work has also been performed to reduce the energy used by shared register file in embedded VLIW architectures [29]. The low power approaches mentioned above are counter-intuitive with thermal management in that the approaches can reduce overall leakage power by scheduling instructions to maximize the time period during which FUs are in a low power mode, as this would increase the power density of some FUs, generating possible hotspots. Recently, the work in [21] applied a load balancing heuristic (not using thermal simulation) to the instruction binding to reduce the peak temperature of FUs. A variety of temperature-aware instruction scheduling techniques have been proposed for clustered architectures with additional hardware supports [19], [20]. On the other side, some work has been done on peak temperature reduction at high level synthesis of ASIC hardware design [25], [26].

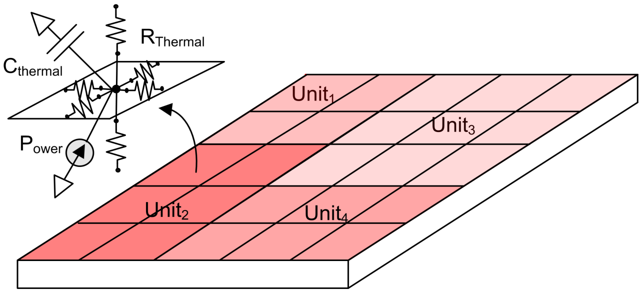

Fig. 4. A conceptual view of thermal equivalent circuit we used.

IV. The Proposed Temperature Control Techniques

The first technique (TempIB) is based on a greedy algorithm that tries to rebind instructions to the coolest possible FUs in each execution of instruction words. The second (fast) technique (TempIB-f), which is based on TempIB, makes use of additional information in order to reduce the number of thermal simulations needed. Note that one of the main aspect in the introduction of thermal-awareness in VLIW compilers is the thermal simulator. We therefore proceed first explaining in detail how our thermal simulator works.

A. Thermal Simulator

In order to have a consistent thermal-aware design flow, a suitable thermal model is needed. On one side it should be accurate enough and on the other side it should be computationally efficient. The thermal model used in this case is based on the known duality between electricity and thermal flow [15] and is based on the model developed by Skandron, *et al.* [10]. Some changes are made from their model as they only consider one type of package (CBGA) with a specific heat sink. In our model, the user can choose a package from a library of different packages so that the equivalent thermal model is generated according to the chosen package. The primary and secondary heat flows will depend on the package type selected. In case of a CBGA package, the primary flow will dissipate heat through the heat sink and the secondary flow will propagate heat through pins of the chip to the PCB.

A thermal mesh is generated on top of the given floorplan, as shown in Fig. 4. The size of each thermal cell is established by the user. A finer mesh will yield a more precise result while taking a longer computational time, whereas a coarser mesh

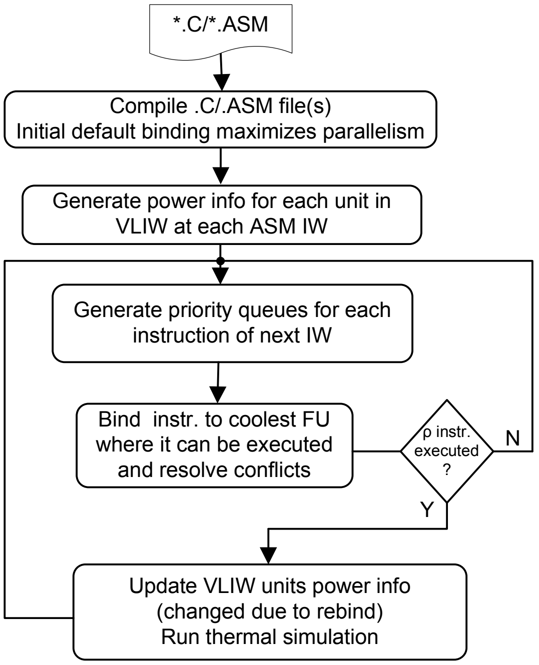
.C/.ASM
Compile .C/.ASM file(s)
Initial default binding maximizes parallelism
Generate power info for each unit in VLIW at each ASM IW
Generate priority queues for each instruction of next IW
Bind instr. to coolest FU where it can be executed and resolve conflicts
ρ instr. executed ?
N
Y
Update VLIW units power info (changed due to rebind)
Run thermal simulation

Fig. 5. Flow graph of our proposed temperature-aware instruction binding technique

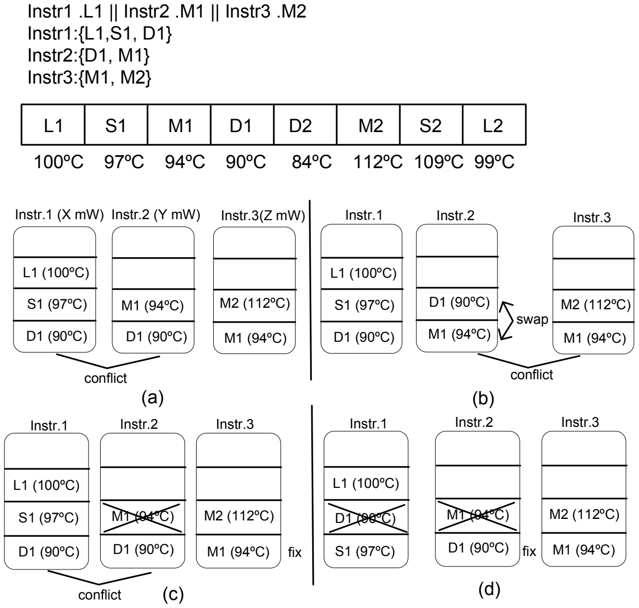

Fig. 6. An example showing the instruction binding by TempIB: (a) Initial priority queue; (b) Binding conflicts; (c) Binding conflicts after FU-swapping; (d) Final bind.

will provide a less accurate result while being faster. Each cell consists of 6 resistors, a capacitor and a current source. The thermal capacitor models the transient behavior of the heat flow and the current source of the generated heat. The resistors in the X-Y plane model the 2-D heat flow on the X-Y plane while the resistors in the Z axis model the heat flow through the primary and secondary flows. The thermal resistors are proportional to their length in the direction of the heat transfer and inversely proportional to its heat transfer surface area and thermal conductivity as given by: $R_{thermal} = L/(k \cdot A)$. On the other hand the thermal capacitor is proportional to the area and the thickness: $C = c_p \cdot \tau \cdot L \cdot A$, where c_p is the specific heat and τ the specific density of the material. The thermal resistance duality allows to solve the heat transfer problem in an analogous manner to electric circuit problems, using the equivalent thermal resistance network, where temperature T is equivalent to the voltage and the heat conduction Q is equivalent to the current. Therefore $\Delta T = Q \cdot R$.

The thermal simulation starts once the equivalent thermal model is generated. A power profile for each unit in the system is passed to the model and the temperature is computed for each thermal cell on each time step. Finite difference equation is used for this propose in order to speed up simulation times. The temperature of each neighboring cell is updated at every time step. The computational time step needs to be small enough so that the heat cannot transfer to the neighboring cell in one time step.

B. Temperature-aware Instruction Binding Technique

Instruction Word (IW) is a grouping of instructions in a single memory location that can be read at the same cycle step by the processor. Our temperature-aware instruction binding technique, TempIB, is based on a greedy algorithm that tries to bind the instructions in *IW* to the coolest possible FUs. Fig. 5 shows the flow of TempIB. It starts from any thermal-unaware binding produced by an initial compilation of the C or assembly code. In this first compilation, the compiler tries to maximize the parallelism involved in the source code by packing as many instructions as possible in parallel. (In the case of the TI's C6x family, up to 8 instructions can be executed in parallel [9].) After this step, the power profile of each FU is generated. The profile depends on which instruction the FU executes. Once this is performed, TempIB will be applied.

The main steps involved in this algorithm are as follows (See Fig. 7.) and an example can be seen in Fig. 6:

Step 1: For each instruction a priority queue is created, sorting the FUs on which the instruction can be executed from minimum to maximum temperatures as shown in Fig. 6(a), as well as considering the power consumption of the instruction in order to estimate the temperature of the FUs once the instruction has been executed[1].

Step 2: In case when there are no conflicts[2] the instructions are bound to the FUs at the bottom of the queue. However, when there is a conflict this needs to be resolved as shown in Fig. 6(b). *Instr.1* and *Instr.2* try to be bound to D1, which is the coolest FU to which the two instructions can be bound. As only one instruction can be executed on that FU, we look at the next FU in the priority queue and the coolest unit of the second element of the queue is swapped by the conflicting FU in the queue. In this case the FUs in *Instr.2* are swapped as M1 is cooler than S1. After this step it is checked again for conflicts until no more conflicts exist. Every time an instruction is

[1] Not all instructions consume the same amount of power

[2] Two instructions do not try to be executed on the same FU.

```
TempIB: Temperature-aware Instruction Binding($\mathcal{F}$,$M$,ρ,$C$,$\mathcal{L}$)
/* $\mathcal{F}$: input floorplan of VLIW processor
   $M$: $n \times n$ thermal mesh
   ρ: thermal simulation interval
   $C$: *.c or *.asm code*/
   $L$: power library of each asm instruction */
• Compile *.asm or *.c file, generating maximum parallelism;
• Generate power profile for each unit $IW$;
• Set $count = 0$;
foreach ($IW$ in $C$) /* rebind instructions */
      /* Step 1 */
      • Create priority queue for each instruction of the FUs
        where they can be executed;
      /* Step 2 */
      • Resolve conflicts by analyzing next FU in the
        queue and the power of the conflicting instructions;
      • $count = count + 1$;
      if ($count$ = ρ)
            • Regenerate power profile of each FU;
            • Rerun thermal simulation;
            • $count = 0$;
      endif;
endfor;
return $C$;
```

Fig. 7. A summary of TempIB.

bound to an FU it is deleted from the priority queue of the rest of the instructions as shown in Fig. 6(c). This algorithm guarantees that each instruction is always bound to the coolest possible unit in each IW.

A new thermal simulation is performed after ρ IWs are bound, where ρ is a (positive integer) parameter set by the user. The smaller ρ is the more precise the results will be, but the longer the run time will be. On the other hand, a large ρ will decrease the accuracy as thermal data is needed to perform the binding, but the technique will run faster. The best results will be achieved if a thermal simulation is performed every time when each of IWs are bound, as an accurate information of the temperature of each FU is available.

C. Enhanced Instruction Binding Heuristic

Note that the most time consuming part of TempIB is the thermal simulation (i.e., *if-block* in Fig. 7), which takes around 95% of the entire run time in practice. However, to get the best results, TempIB needs (ideally) to perform a thermal simulation after the binding of each new IW to use the previously refreshed new power profile of each FU. This would guarantee that the temperature of each FU is up to date after each rebinding is performed. However, it leads to a computationally expensive run time as every thermal simulation has to iteratively update the temperature of each thermal node. On the other hand, it looks obvious not to perform a thermal simulation after each IW as one single IW does not increase the FUs' temperature too much. We found in the experiments that the minimum number of IWs that can be bound before a new thermal simulation was applied while maintaining the accuracy of thermal estimation is around 50 (estimated performing some initial tests).

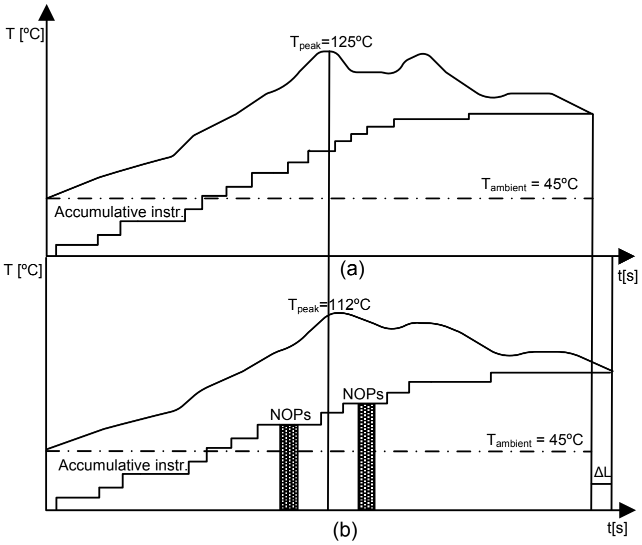

Fig. 8. Example showing the application of TempNOP: (a) Initial temperature curve; (b) Temperature curve after applying TempNOP.

D. Temperature-aware NOP Insertion Technique

We describe what we called temperature-aware NOP insertion technique (TempNOP), which is quite natural to be included as a postprocessing to the conventional compilers which bind instructions for maximizing performance. We will compare and discuss, in the experimentation section, the results produced by TempIB and TempIB-f with that by TempNOP.

```
TempNOP: Temperature-aware NOP Insertion($\mathcal{F}$,$M$,$C$)
/* $\mathcal{F}$: input floorplan of VLIW processor
   $M$: $n \times n$ mesh
   $C$: *.c or *.asm code*/
/* Step 1 */
• Compile *.asm or *.c file; /* maximize parallelism */
• Generate power info. for each unit for each step;
/* Step 2 */
while (Time constraint met with $\Delta L$)
      • Apply thermal simulation to $\mathcal{F}$ and obtain thermal
        signature of FUs;
      • Find longest instruction sequence on the hottest FU;
      • Insert X NOPs in the middle of the instruction sequence;
endwhile;
return $\mathcal{F}$;
```

Fig. 9. A summary of temperature-aware NOP insertion technique TempNOP.

TempNOP is based on a very intuitive assumption that in order to cool down a unit, the unit needs to "rest" in order to stop the temperature build-up. An example can be seen in Fig. 8. Fig. 8(a) shows the initial temperature build-up of an FU with time as well as the accumulative instruction executions on it. On the other hand, Fig. 8(b) shows the temperature build-up after TempNOP is run, indicating that NOPs have been inserted and how this influences on the total execution time overhead, which is given by ΔL in the figure.

Fig. 9 shows the pseudo code of TempNOP. Temperature is additive in nature and this temperature reduction technique considers the entire thermal history of each unit. Once the original C or assembly code is compiled and optimized for parallelism, TempNOP performs an initial thermal simulation of the entire system (i.e., Step 1 in Fig. 9). A thermal signature is obtained for each FU of the processor. The thermal signature is basically the thermal history of each FU. These signatures are analyzed and the highest peak temperature is then singled out in order to be treated. A hard timing constraint is also given by the user not allowing the code to be executed in more than a given amount of time (ΔL) (i.e., Step 2 in Fig. 9). As the initial code is optimized for performance only, a worse result can be achieved by the new instruction assignment. For each peak temperature, a set of NOPs is inserted in the code. This allows the unit to cool down. The number of NOPs inserted depends on the peak temperature, its duration, and by the size of the hotspot compared to the next largest temperature peak as inserting too many NOPs would decrease the temperature of the current hotspot too much not allowing room to reduce the temperature of the next hotspot. TempNOP searches from the first to the last instruction that contributes to the peak temperature of the FU. It then inserts the NOPs in the middle of the longest consecutive instruction sequence. We assume this is the point where the largest temperature build-up is most likely to happen.

V. Experimental Results

First, we describe the experimental setup for the generation of initial floorplan and our proposed thermal-aware design flow. Then, we show a set of comprehensive results obtained together with explanations on the implication and analysis of the data.

A. Experimental Setup

To validate the temperature reduction and control techniques proposed, the Texas Instruments C6200 architecture was chosen [9], as it is one of the most popular VLIW architectures. The disadvantage of using a commercial processor is the lack of detailed information provided by the vendor. Some assumptions had to be made to bridge this lack of information like the exact floorplan and the power consumption of each unit. Fig. 1(a) shows the DSP floorplan used for the experiments based on the TI's logic diagrams from their datasheet [9]. On the other hand, the power consumption values for each unit in the design are based on [16], which uses a look up table approach for each executed instruction, where the power consumed by each unit when executing an instruction is given in a power library. The power values used were validated using [17].

The benchmarks tested were also downloaded from the Texas Instrument's web page [9]. Six of them are DSP applications while the remaining are image processing benchmarks. We extracted the representative assembly code from the benchmarks. Table II shows the name of the benchmarks, their number of instruction words, and the parallelism involved in the benchmarks. This is obtained by going through all the *IWs* and checking how many instructions of the total maximum of 8 instructions (the maximum number of instructions per *IW* is 8 for the C6X TI DSP family) that can be executed in parallel are really executed, as indicated by:

$$Parallelism(\%) = \frac{tot\_exe\_instructions}{(tot\_IWs)\cdot(max\_instr\_per\_IW)} \times 100 \quad (1)$$

TABLE II

Benchmark description used at the experimental section

Benchmark	#IWs	#Instructions	Parallelism (%)
DSP_BLK_MOVE	54	120	27.78
DSP_FIR_LMS2	95	98	12.89
IMG_BOUNDARY	104	109	13.10
MPEG2_VLD	107	457	53.39
DSP_MUL32	147	150	12.76
DSP_FLTOQ15	148	225	19.00
DSP_AUTOCOR	179	395	27.58
IMG_CONV3X3	196	420	26.79
DSP_FIR_CPLX	221	295	16.69
IMG_PIX_SAT	241	411	21.32
IMG_SOBEL	371	489	16.48
IMG_PIX_EXPAND	454	691	19.03

B. Experimental Results

Table III shows the results of the temperature reduction techniques for the different benchmarks described in Table II. The first two columns show the benchmark names and the peak temperature (T_{init}) produced by a temperature-unaware code compilation. Columns 3 to 5 show the peak temperature (T_{peak}) and the amount of decrease of temperature in % over T_{init} by our TempIB, and the run time where our thermal simulation is performed whenever each *IW* is rebound. On the other hand, columns 6 to 8 show the results by TempIB, performing the thermal simulation at rebinding of every 50 *IWs*. Similarly, columns 9 to 11 show the results by TempIB-f with the thermal simulation interval of 50 *IWs*. The last columns from 12 to 15 show the results by TempNOP, allowing up to 15% timing degradation. The best TempIB (i.e., performing a thermal simulation after each rebinding) reduces the peak temperature by 10.91% on average, but takes about 132 seconds to execute. TempIB which calls a thermal simulation after 50 *IWs* on the other hand reduces the peak temperature by 8.37%, but runs around 40 times faster. However, we can see that TempIB-f reduces the temperature by 10.19%, almost the same as the best case where a thermal simulation is performed in each rebinding step, and is around 14 times faster. TempNOP on the other hand can only reduce the peak temperature by 4.03% even sacrificing 15% of performance.

VI. Conclusions

Temperature is increasingly becoming important in VLSI circuits and needs to be addressed as a separate factor when designing ICs. This work addressed a new problem of temperature-aware instruction binding and proposed two effective techniques, TempIB and TempIB-f. In summary, TempIB was able to reduce the peak temperature of FUs by up to 13.82% and TempIB-f by up to 12.70%. The effectiveness of the techniques was also validated by comparing them with an NOP insertion method, which could only reduce the peak temperature by up to 8.23% even with a timing penalty of 15%.

TABLE III

A COMPARISON OF PEAK TEMPERATURE AND RUN TIME OF THREE TEMPERATURE-AWARE INSTRUCTION BINDING TECHNIQUES TempIB, TempIB-f AND TempNOP (ρ = THERMAL SIMULATION EVERY *n* IWS).

		TempIB (ρ = 1 *IWs*)			TempIB (ρ = 50 *IWs*)			TempIB-f (ρ = 50 *IWs*)			TempNOP (ΔL = 15%)		
	T_{init}	T_{peak}	ΔT%)	run (s)	T_{peak}	ΔT (%)	run (s)	T_{peak}	ΔT (%)	run (s)	T_{peak}	ΔT (%)	run (s)
DSP_BLK_MOVE	57.30	49.38	13.82	9.42	51.67	9.83	0.25	50.55	11.78	1.9	55.21	3.65	1.39
DSP_FIR_LMS2	52.39	48.15	8.09	29.53	49.39	5.73	0.28	48.39	7.64	3.36	51.49	1.72	4.78
IMG_BOUNDARY	52.48	47.9	8.73	35.15	49.19	6.27	0.93	48.19	8.17	4.05	51.33	2.19	5.88
MPEG2_VLD	66.46	57.39	13.65	44.83	58.85	11.45	1.11	57.96	12.79	4.48	63.95	3.78	4.26
DSP_MUL32	52.36	48.19	7.96	65.31	48.78	6.84	1.12	48.39	7.58	5.85	51.38	1.87	11.64
DSP_FLTOQ15	57.36	50.36	12.20	87.17	51.75	9.78	1.57	50.65	11.70	5.92	54.21	5.49	10.99
DSP_AUTOCOR	60.52	53.24	12.03	95.21	54.79	9.47	2.77	53.23	12.05	8.53	57.38	5.19	17.3
IMG_CONV3X3	61.87	53.77	13.09	125.29	54.04	12.66	3.09	54.01	12.70	9.01	56.78	8.23	21.58
DSP_FIR_CPLX	58.35	51.19	12.27	153.56	53.19	8.84	3.02	51.88	11.09	10.17	56.08	3.89	36.69
IMG_PIX_SAT	54.93	50.96	7.23	201.41	51.99	5.35	3.79	51.17	6.85	13.73	54.87	0.11	50.21
IMG_SOBEL	59.38	52.13	12.21	309.11	53.88	9.26	9.87	52.87	10.96	15.14	54.93	7.49	35.53
IMG_PIX_EXPAND	57.48	51.97	9.59	429.42	54.63	4.96	10.83	52.31	8.99	28.16	54.78	4.70	67.39
Avg.	57.57	51.22	**10.91**	132.12	52.68	**8.37**	3.22	51.63	**10.19**	9.19	55.20	**4.03**	22.30

REFERENCES

[1] National Semiconductor, *Understanding Integrated Circuit Package Power Capabilities,* www.national.com, pp. 291-301, April, 2000.

[2] F. Fallah and M. Pedram, “Standby and active leakage current control and minimization of CMOS VLSI circuits,” *IEICE Transactions on Electronics,* Vol. E88-C, No. 4, pp. 509-519, 2005.

[3] S. Borkar, “Design challenges of technology scaling,” *IEEE Micro,* Vol. 19, No.4, pp. 23-29, 1999.

[4] R. Mukherjee and S.O. Memik, “Systematic temperature sensor allocation and placement for microprocessors,” *Design Automation Conference (DAC)*, pp. 542-547, 2006.

[5] D. P. Pham, “The design and implementation of a first-generation CELL processor: a multi-core supercomputer SoC,” *IEEE Custom Integrated Circuits Conference,* pp. 45-50, 2005.

[6] C.H. Tsai and S.M. Kang, “Standard Cell Placement for Even On-Chip Thermal Distribution,” *ISPD* , pp. 179-185, 1999.

[7] C.C. Chu and D.F. Wong, “A Matrix Synthesis Approach to Thermal Placement ,” *IEEE TCAD,* Vol.17, No11, pp. 1166-1174, 1998.

[8] A. Gupta et. Al, “Flooplan Driven Leakage Power Aware IP-Based SoC Design Space Exploration,” *CODES-ISS,* pp. 118-123, 2006.

[9] *www.ti.com*, Texas Instruments.

[10] W. Huang, M. R. Stan, K. Skadron, and K. Sankaranarayanan, “Compact thermal modeling for temperature-aware design,” *Design Automation Conference (DAC)*, pp. 878-883, 2004.

[11] W. Huang, J. Renau, S-M Yoo, and J. Torellas, “A framework for dynamic energy efficiency and temperature management,” *International Symposium on Microarchitecture,* pp. 202-213, 2000.

[12] K. Skadron, M. R. Stan, W. Huang, S. Velusamy, K. Sankaranarayanan, and D. Tarjan, “Temperature-aware microarchitecture: Modeling and implementation,” *ACM Transactions on Architecture and Code Optimization (TACO)*, Vol. 1, No. 1, pp. 94-125, 2004.

[13] C. H. Lim, W. Daasch and G. Cai, “A thermal-aware superscalar microprocessor,” *International Symposium on Quality of Electronic Design (ISQED)*, pp. 517-522, 2002.

[14] K. Skadron *et al.*, *Temperature-aware microarchitecture: extended discussion and results,* Tech. Report CS-2003-08, University of Virginia, CS department, Apr. 2003.

[15] Y. A. Cengel, *Introduction to thermodynamics and heat transfer,* Mc-Graw Hill, ISBN 0070114986, 1996.

[16] V. Tiwari, S. Malik and A. Wolfe, “Power analysis of embedded software: A first step towards software power minimization,” *IEEE Transactions on Very Large Scale Integration (VLSI) Systems,*, Vol. 2, No.4, pp.437-445, 1994.

[17] N. Julien, J. Laurent, E. Senn, and E. Martin, “Power consumption modeling and characterization of the TI C6201,” *IEEE Micro*, Vol. 23, No. 5, pp. 10-49, 2003.

[18] D. Brooks and M. Martonosi, “Dynamic thermal management for high-performance microprocessors,” *International Symposium on High-Performance Computer Architecture (HPCA)*, pp. 171-182, 2001.

[19] P. Chaparro , J.Gonzalez and A. Gonzalez, “Thermal-aware clustered microarchitectures, *International Conference on Computer Design (ICCD),* pp. 48-53, 2004.

[20] S. Haga, N. Reeves, R. Barua and D. Marculescu, “Dynamic functional unit assignment for low power,” *The Journal of Supercomputing*, Vol.31, No.1, pp. 47-62, 2005.

[21] M. Mutyam, F. Li, N. Vijaykrishnan, M. Kandemir, and M. J. Irwin, “Compiler-directed thermal management for VLIW functional units,” *Conference on Languages, Compilers, and Tools for Embedded Systems (LCTES),*, pp. 163-172, 2006.

[22] H. S. Kim, N. Vijaykrishnan, M. Kandemir, and M.J. Irwin, “Adapting instruction level parallelism for optimizing leakage in VLIW architectures,” *Conference on Languages, Compilers, and Tools for Embedded Systems (LCTES),* pp. 275-283, 2003.

[23] H. Yun and J. Kim, “Power-aware modulo scheduling for high-performance VLIW processors,” *International Symposium on Low Power Electronics and Design (ISLPED),* pp. 40-45, 2001.

[24] W. Zhang, N. Vijaykrishnan, M. Kandemir, and M.J. Irwin”, “Exploiting VLIW schedule slacks for dynamic and leakage energy reduction,” *International Symposium on Microarchitecture,* pp. 102-113, 2001.

[25] R. Mukherjee, S. O. Memik, and G. Memik, “Temperature-aware resource allocation and binding in high-level synthesis,” *Design Automation Conference (DAC),* pp. 196-201, 2005.

[26] P. Lim and T. Kim, “Thermal-aware high-level synthesis based on network flow method,” *International Conference on HW/SW Co-design (CODES),* pp. 124-129, 2006.

[27] O. Coudert, J. Cong, S. Malik, and M. Sarrafzadeh, “Incremental CAD,” *International Conference on Computer-Aided Design (ICCAD),* pp. 236-243, 2000.

[28] Z. Gu, J. Wang, R. P. Dick, and H. Zhou, “Incremental exploration of the combined physical and behavioral design space,” *Design Automation Conference (DAC),*, pp. 208-213, 2005.

[29] J. L. Ayala, D. Atienza, P. Raghavan, M. Lopez-Vallejo, F. Catthoor, and D. Verkest, “Energy-aware compilation and hardware design for VLIW embedded systems,” *International Journal of Embedded Systems (IJES),* to appear in Dec. 2006.