Published in IET Computers & Digital Techniques
Received on 31st March 2009
Revised on 30th July 2009
doi: 10.1049/iet-cdt.2009.0041

ISSN 1751-8601

Exploration and implementation of a highly efficient processor element for multimedia and signal processing domains

M. Lai L. Gao Z. Wang
School of Computer National University of Defense Technology, Changsha, People's Republic of China
E-mail: mingchelai@nudt.edu.cn

Abstract: The exploration and design process of highly efficient processor element for multimedia and signal processing domains is presented in this study. With the introduction of synchronous data-transfer architecture for high-performance embedded applications, the effectively exploring the exponential-size architectural design spaces by detailed simulation is intractable. The authors attack this via an automated approach. At first, its cost model is built to achieve fast and accurate estimation with the characteristic of scalability. Then, the hierarchical design space exploration methodology involving heuristic-based local process and analytical global optimisation step is proposed to achieve the approximate optimum with short time-to-market. For target domains, our proposed method arrives at better optimised results within only 25 h when compared with other methods. A System on Chip (SoC) involving the optimised processor element has been implemented in 0.13 μm complementary metal oxide semiconductor (CMOS) process and the experimental results show that our processor element outperforms TMS320C64 series and does the obvious acceleration to multimedia applications in SoC system.

1 Introduction

In multimedia and signal processing domains, with the evolvement of standard algorithms and the increasing magnitude of computing requirements, many cost-sensitive embedded systems pursue the high performance to realise real-time processing. Up to now, there have been many attempts by digital signal processors (DSP), field programmable gate arrays (FPGA) or application specific integrated circuits (ASIC). However, the continuously changing landscape and the strict performance requirements favour the future processors to be programmable, high performance and cost-effective. Recently, with the advancement of very-large-scale integration technology, many processor solutions [1, 2] composed of a few of cores integrated on a single chip have been proposed to deal with power issues dominating deep sub-micron technologies and make it easy to exploit thread-level parallelism to provide significant energy and performance advantages.

Most embedded multi-cores use the processor element following superscalar or very long instruction word (VLIW) architecture. But the high-performance and low-cost requirements cannot be solved with these cores which rely on the increasing clock rates and large caches, both of which consume energy inefficiently. Superscalar processor adopts the complex hardware to exploit the instruction level parallelism and exposes the disadvantages of weak computing ability, low-energy efficiency and poor scalability [3]. The VLIW relies on compile-time detection of parallelism, but its complexities of decoder or data bypasses which increase exponentially with the number of function units are likely to result in a poor proportion of resources contributed to computation [3, 4]. The popular cores in the competitive embedded products should follow a flexible computing architecture which exploits the inherent parallelism of programs and pursues the high utility ratio of transistors to improve energy efficiency. The computing core in this paper moves into the synchronous data transport architecture (SDTA), which is a variation of the traditional transport triggered architecture [5]. The main advantages are centralised on its simpler control mechanism, higher hardware utility ratio and finer-grained programming model.

Using the SDTA architecture, it is then extremely important to design the highly efficient processors, achieving higher performance while consuming minimal energy and silicon area. Up to now, there existed lots of prior works which used the design space exploration to organise balanced processor. The challenge was to explore a great number of choices for target domains within short time-to-market. A first approach was to prune the explored design space [6]. It could yield significant simulation time reductions but was hard to adopt the proper strategy to get close to global optimum. Second, the heuristic-based approach included hill climbing [7], one-parameter-at-a-time [8], tabu search [9] and genetic search methods [10]. The main limitation was that they were prone to getting stuck in local minima, and how to improve these methods in conjunction with other approaches to get close to optimal point was worthy to be considered. Third, the statistical simulation approach [11, 12] reduced the number of simulated instructions. It collected program statistics through trace-driven simulation, but a great diversity of program behaviours could not be reflected during the synthetic instruction trace. In addition, the analytical optimisation approaches [13–15] adopted some analytical-based process to output the Pareto-optimal processors. It used the relatively accurate analytical models to obtain the optimum in a short period, but most ones were limited to the superscalar to our knowledge.

In order to explore and design the highly efficient processor element for target domains, this paper develops an automated optimisation approach which obtains the Pareto-optimum with respect to performance-energy under area constraints. The main contributions of this paper are concentrated on three aspects. Firstly, the cost analytical model of processor element with the advantages of flexibility and high efficiency is proposed to meet the precision requirements and suits for the application-specific exploration. Secondly, the optimal processor elements are derived by a hierarchical optimisation process effectively. To improve the efficiency, the hierarchical optimisation approach explores Pareto-optimal cores with respect to performance energy in different area intervals at first, and then proposes an analytical method with trace simulation to optimise the overall processor. The proposed automated approach arrives at approximate optimum only within 25 h and obviously outperforms other design space pruning method or genetic algorithm. Finally, for the representative benchmarks of multimedia domains [16, 17], the optimal processor element through our approach has been implemented in 0.13 μm 1P8M CMOS process. The experimental results show that our optimised processor element with the advantage of low power consumption outperforms TMS320C64 in performance and yields obvious acceleration to target multimedia benchmarks.

2 Processor element architecture

The proposed processor element is based upon the SDTA architecture, which is a variation of the traditional transfer triggered architecture (TTA) [5]. It inherits the advantages of exploiting multiple-level parallelisms but modifies the shortcomings of TTA mainly on three aspects. The unreasonable pipeline structure is improved firstly. In our processor element, most simple operations are triggered by data transfers within transfer stage while the executions of other complex operations are arranged in the next cycle. In this way, many computing results are immediately available by the next-slot instructions and it is easy to reduce the program runtime without any influence to the frequency by balancing the overhead of each cycle. Secondly, the special bypass and branch-decoder mechanisms are used to reduce branch penalties of conditional jumps, unconditional jumps and returns to two cycles in contrast with four-cycle stalls of TTA, and it could avoid great performance losses. Thirdly, the multimedia extension techniques are introduced, referring to Intel's MMX, SSE1 and so on. [18]. They exploit the available data level parallelism in multimedia programs for high performance and then provide less memory bandwidth requirements.

The major difference between SDTA architecture and traditional VLIW one is the way that the operation is executed. In our programming model, the program only specifies the data transports on the network. Operations occur as a 'side-effect'. In SDTA architecture, each function unit has one or several operator registers, result registers but only one trigger register. Data transferred to trigger register will trigger the corresponding unit to work, where different operations are indicated by slot codes. The advantages of the proposed architecture during exploiting parallelism are summarised as follows. Firstly, the processor elements are configured with an abundance of function units, and many custom resources are connected with high-bandwidth transport network to meet high-performance requirement. Meanwhile, simplifying instruction decoder logic, issue logic and data paths results in a high utility ratio of silicon resources, thus the energy consumption per operation is decreased. Secondly, the distributed operand and result registers are available to store variables. The distributed framework with scalability characteristic reduces the pressure of multi-ports register files and improves the performance effectively. Thirdly, the decoupled architecture among the units and network is more suitable for exploiting deep pipeline to make full use of resources. With multimedia extensions, it supports both the single instruction stream, multiple data streams (SIMD) and multiple instruction streams, multiple data streams (MIMD). Fourthly, the data-triggered property of programming model is more flexible and fine-grained than traditional operation-triggered ones, for example VLIW. The programming model which allows the transfer parallelism among several traditional-dependent operations is suitable for more efficient schedule in instruction level as well as in data level.

Then, we can customise the processor element of the above architecture to meet the requirement of target domains, for example the multimedia and signal processing applications. To decrease the time-to-market coupled with increasing design space and keep the software-based compatibility with less development effort, the processor element needs to be modelled using a uniform architecture description language [19] which exists above the register transfer level by ignoring details. Using architecture description files, the instruction mapping and the parallel schedule may be performed by software-based toolkits, and the hardware-based toolkits may generate the Verilog descriptions of processor elements. Fig. 1 illustrates the general architecture of SDTA processor element, the oval boxes denote function units, dotted boxes denote storages, bold edges are pipeline edges and dotted edges are data-transfer edges. A data-transfer edge transfers data between units and storages. A path from a root node (e.g. fetch1) to a leaf node is called a pipeline path (e.g. {fetch1, fetch2, decode, transfer}). Our processor element is composed of a great number of computing resources, involving arithmetic units, float units, cordic unit [20], register files, load/store units, io/control unit and channel unit [21], which issues the direct memory access (DMA) commands with unique identifiers to DMA engine to transfer valid data between local storage and other memory spaces. Then, the right dashed box represents the memory system including the instruction cache, local storage and DMA engine [22], which receives and arbitrates DMA commands from channel unit, instruction cache or remote agents, with responsibility for transferring data between different memory spaces. As all known, many multimedia programs require a bandwidth-oriented memory system. However, the conventional cache-based memory hierarchies organised with limited cache memories which lack of the efficient prefetch always lead to a poor performance because of poor temporal locality. The introduction of local storage may simplify the hardware overhead of caches, and its decoupled memory access allows software to schedule data transfers in parallel with core computation thereby overcoming memory latencies and achieving high memory bandwidth.

3 Modelling area and power characteristics

During the exploration of a huge number of processor elements, the cost estimation approach is extremely important for determining optimal choices and its key characteristics should be emphasised to be accurate, highly efficient and scalable. Up to now, extensive research efforts have been put to develop efficient power estimation methods at all levels of design process but cannot be applied to exploration process. The point of our work is not to compete with other tools, but rather to develop a cost model with the characteristics of accuracy, high-efficiency and scalability for space exploration.

3.1 Processor area model

With each elementary component, there is an associated area cost. As shown in (1), the area estimation of entire processor is composed of individual component areas. Most of the component areas are derived from cost library except for transfer network and control logic. The area of transfer network is the area sum of sockets and busses that are implemented by AND gates and multiplexers, respectively. The control logic includes the instruction fetch and decoder components. The area of instruction fetch component is synthesised to be little and in proportion to the instruction width. The area of decoder component is closely related to transfer network and function unit number, and it can be derived using the analytical methods similar with the power consumption model given below

$$A_{\text{PE}} = A_{\text{fu}} + A_{\text{RF}} + A_{\text{cntrl}} + A_{\text{network}} + A_{\text{icache}} + A_{\text{dma}} + A_{\text{local}} \tag{1}$$

3.2 Power consumption model

With the modularity characteristic of SDTA element, the mainly modelled components fall into several categories, including transfer network, register files, function units,

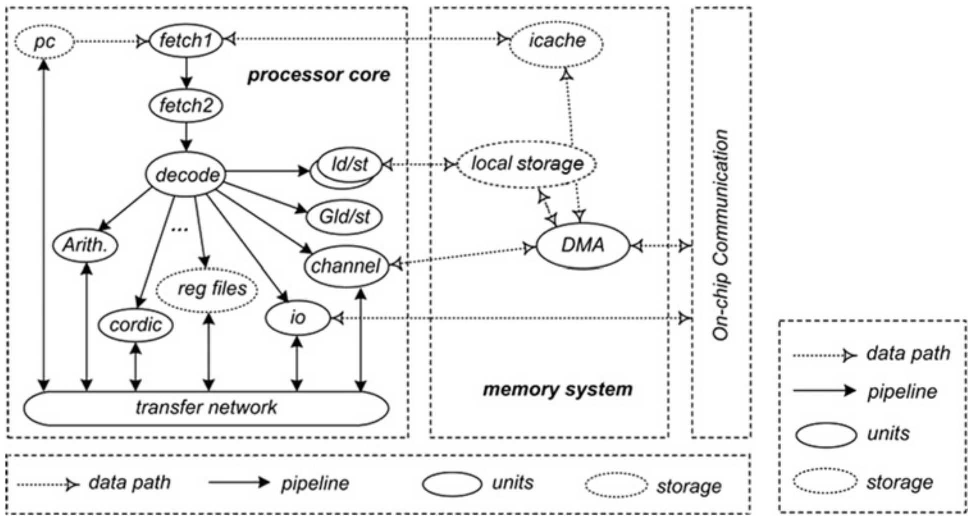

Figure 1 *General architecture of our processor element*

decoder units, on-chip memories and so on. Based on the distinct underlying structures of each component, the hybrid power model is developed as follows.

Every function unit consumes the power as (2), which is expressed as the sum of dynamic power and leakage current power consumption $P_{\mathrm{fu}(i),\mathrm{s}}$. The dynamic energy is expressed as the sum of all the active operation energy, which is the product of operation execution times f_j and dynamic energy per operation $E_{\mathrm{fu}(i),j}$. The average power consumption is given by $P = E/\mathrm{Time}_{\mathrm{exec}}$, where $\mathrm{Time}_{\mathrm{exec}}$ denotes the program runtime. In addition, the power consumption for register file is estimated using (3), where $E_{\mathrm{RP}(i)\mathrm{WP}(j)R(m)W(n)}$ is the average dynamic energy when m reads and n writes are executed in parallel, and $f_{\mathrm{RP}(i)\mathrm{WP}(j)R(m)W(n)}$ is the number of access times

$$P_{\mathrm{fu}(i)} = \sum \frac{E_{\mathrm{fu}(i),j} \times f_j}{\mathrm{Time}_{\mathrm{exec}} + P_{\mathrm{fu}(i),\mathrm{s}}} \tag{2}$$

$$P_{\mathrm{RF}} = \sum \frac{E_{\mathrm{RP}(i)\mathrm{WP}(j)R(m)W(n),\mathrm{size}} \times f_{\mathrm{RP}(i)\mathrm{WP}(j)R(m)W(n),\mathrm{size}}}{\mathrm{Time}_{\mathrm{exec}} + P_{\mathrm{RP}(i)\mathrm{WP}(j),\mathrm{size,s}}} \tag{3}$$

The transfer network acts as the backbone of processor element and is composed of three basic units involving busses, input and output sockets. In general, these basic units are implemented by AND or multiplexers which are classified into bit-independent units [23]. Therefore the transfer network could be partitioned into basic units with small lookup table thereby achieving the accurate results and improving the efficiency. The transfer network consumes power consumption as (4) and it is the power sum of sockets and busses. For each socket, their dynamic energy E_{sk}, and the leakage current power P_{sk}, are derived from cost library, whereas the input transition is collected from simulation. For busses, some switch tables for AND operations are used for power estimation. With the collected signal transitions, the ith bus uses its own capacitive load u as the index to search for the information in cost library for computing its dynamic energy

$$\begin{aligned} P_{\mathrm{network}} &= \frac{(\Sigma E_{\mathrm{sk\_}i} + \Sigma E_{\mathrm{sk\_o}} + \Sigma E_{\mathrm{bus}})}{\mathrm{Time}_{\mathrm{exec}} + (\Sigma P_{\mathrm{sk\_}i} + \Sigma P_{\mathrm{sk\_o}} + \Sigma P_{\mathrm{bus}})} \\ &= \frac{(\Sigma E_{\mathrm{mux,2}} + \Sigma E_{\mathrm{mux,p}} + \Sigma E_{\mathrm{AND,u}})}{\mathrm{Time}_{\mathrm{exec}} + (\Sigma P_{\mathrm{mux,2,s}} + \Sigma P_{\mathrm{mux\,p,s}} + \Sigma P_{\mathrm{AND,u,s}})} \\ &\quad (p = 2, \ldots, 10;\ u = 10, \ldots, 50) \end{aligned} \tag{4}$$

The power of decoder exists in (5) and (6). In general, the decoder logic of transfer bus is independent with each other, and thus we measure the total decoding power by a division approach. For a representative decoder logic where u units are connected to the bus, $\mathrm{ET_u}$, $\mathrm{EF_u}$ denote the average decoding power when conditional fields are true or false, respectively. When collecting the successful and failure decoding times of the jth bus associated with M_j units, we introduce a factor λ to adjust the reference. The factor λ is linearly calculated by control bits as shown in (6), where r_i is the number of control bits corresponding to the representative case associated with i units, r_{fu} and r_{reg} are the number of control bits for function units and register files in the estimated cases. For the area estimation of decoder, the similar method is followed, and another parameter A_{u} instead of $\mathrm{ET_u}$ and $\mathrm{EF_u}$ is inbuilt to calculate the parameter A_{decoder}

$$\begin{aligned} P_{\mathrm{decoder}} &= \sum_j \frac{(\mathrm{ET}_{M_j} \times \mathrm{Num}_{j,\mathrm{true}} + \mathrm{EF}_{M_j} \times \mathrm{Num}_{j,\mathrm{false}}) \times \lambda}{\mathrm{Time}_{\mathrm{exec}}} \\ &\quad + \sum_j P_{\mathrm{s},M_j} \times \lambda \end{aligned} \tag{5}$$

$$\lambda = \frac{(\sum r_{\mathrm{fu}} + \sum r_{\mathrm{reg}})}{r_i} \tag{6}$$

The power of on-chip memories are modelled by considering different subcomponents (tag arrays, data arrays and so on). This can be done using PrimePower toolkit [24] in the synthesisable design flow, and the energy consumption of an access on a cache with particular size and associativity is derived. $E_{\mathrm{ic,r}}$, $E_{\mathrm{ic,w}}$ denote the average read and write dynamic energies of instruction cache, $E_{\mathrm{l,r}}$, $E_{\mathrm{l,w}}$ denote the access dynamic energies of local storage, and $P_{\mathrm{ic,s}}$, $P_{\mathrm{l,s}}$ are the leakage current power of instruction cache and local storage, respectively. Then, with the power analysis towards the DMA engine implementation, $E_{\mathrm{dma,miss}}$ denotes the energy of instruction transfers after a miss, $E_{\mathrm{dma,in}}$ and $E_{\mathrm{dma,out}}$ represent the average energies of each inflow and outflow items associated with DMA requests. The power of instruction cache, local storage and DMA engine follow (7)–(9), respectively. Besides the hit in caches, $\mathrm{miss_{ic}}$ represents the miss ratio. $\mathrm{SP_{ic}}$ denotes the dynamic instruction number, $\mathrm{SP_r}$, $\mathrm{SP_w}$ indicate the number of accesses to local storages and $\mathrm{SP_{get}}$, $\mathrm{SP_{put}}$ are the number of inflow and outflow items (see (7)–(9))

$$P_{\mathrm{ic}} = \frac{(\mathrm{SP_{ic}} \times E_{\mathrm{ic,r}} + \mathrm{SP_{ic}} \times \mathrm{miss_{ic}} \times E_{\mathrm{ic,w}})}{\mathrm{Time}_{\mathrm{exec}} + P_{\mathrm{ic,s}}} \tag{7}$$

$$P_{\mathrm{local}} = \frac{((\mathrm{SP_{put}} + \mathrm{SP_r}) \times E_{\mathrm{s,r}} + (\mathrm{SP_{get}} + \mathrm{SP_w}) \times E_{\mathrm{s,w}})}{\mathrm{Time}_{\mathrm{exec}} + P_{\mathrm{local,s}}} \tag{8}$$

$$P_{\mathrm{dma}} = \frac{(E_{\mathrm{dma,in}} \times \mathrm{SP_{get}} + E_{\mathrm{dma,out}} \times \mathrm{SP_{put}} + \mathrm{SP_{ic}} \times \mathrm{miss_{sic}} \times E_{\mathrm{dma,miss}})}{\mathrm{Time}_{\mathrm{exec}} + P_{\mathrm{dma,s}}} \tag{9}$$

3.3 Cost model evaluation

Given the cost library, the results from analytical model are compared with analysis results from tools. We select a wide range of processor choices on the design space and perform the baseline programs. The cost library parameters and the activity information collected by cycle-based simulator are imported into cost model to calculate the estimated area and power. Meanwhile, the RT level descriptions are synthesised using design compiler to gain the area references and the gate-level power analysis by prime power is used to gain power references. Fig. 2 presents the boxplot of error distributions from the power estimations of 100 random validation choices. The error is expressed as |observation-estimation|/observation. Boxplots are graphical displays of data to measure the average, identify the outlier and indicate the distribution dispersion. As shown in Fig. 2, the power model achieves the median errors of 8.1, 7.5, 12.2, 5.2 and 9.1% for each component, respectively, and the box length called inter-quartile range indicates the least error dispersion of transfer network, in contrast with the worst dispersion of decoder. During the exploration, the decoder power model is emphasised to be efficient and sensitive to a wide range of configurations at the expense of accuracy because of the linear estimation from representative cases. For transfer network, combined with its regular underlying structure, the RT level estimation by a division approach is adopted for the more accurate results. Fig. 3 plots the cumulative distribution of the power error, quantifying the number of total estimation results with less than a particular error. Axis x denotes the error and axis y indicates the percentage of validation choices with less than a particular error. Seventy per cent and Ninty per cent of the power errors are less than 8.9 and 12.2%, respectively. In addition, the average and maximum area errors are also measured at 4.0 and 6.9% for the same validation choices. The correlation coefficients of area and power on the above sample spaces are calculated to be 0.989 and 0.972, respectively, which argues that our cost model tracks the observed results well [25].

The efficiency of cost model seems to be more important than the absolute accuracy while exploring design space. In practice, the above area and power model are integrated into a simulator, which outputs the cost estimation through collecting the execution information and referring to the in-built cost library. For example, the performance simulator consumes about 2.2 s when performing 1024-points fft program. If the cost estimation option is added, the simulation needs about 2.7 s for the same program. Thus, the above cost model has little influence on the estimation efficiency.

4 Automated optimisation approach

4.1 Overall automated framework

With the parameterised configurations for the architecture in Fig. 1, the new framework made up of analytical cost model and automated exploration process is proposed to optimise and design the SDTA processor element with short time-to-market, pursuing the performance-energy optimal for multimedia and signal processing domains. The automated optimisation methodology to explore and design the SDTA processor is illustrated in Fig. 4, where the two-phase automated process refers to a series of toolkits including an architecture-sensitive compiler, a sequential simulator, a parallel simulator, an explorer and a hardware generator.

In the first phase, the front-end compiler transforms the baseline programs or kernels written in high-level language into sequential codes. By importing it into sequential simulator, the statistical parameters including the type and percentage of operations, the average active registers and the parallelism upper bound are used to roughly determine the initial computing core. Here, according to the statistical parameters, we can also profile complex programs to highlight parallelism segments, whose code will be scheduled effectively by latter back-end compiler. Then, as shown in Fig. 4, the InstrGen toolkit uses its machine description file to generate instruction set profile and the back-end architecture-sensitive compiler schedules the sequential codes into parallel ones. In our back-end compiler, its parser reads the sequential codes and machine description file to produce basic forms of intermediate code, for example the target-machine instructions, the basic blocks and the procedures of program. Next the scheduler begins the control-flow and data-flow analysis, and considers the above statistical parameters to estimate the performance of different schedule measures, such as loop

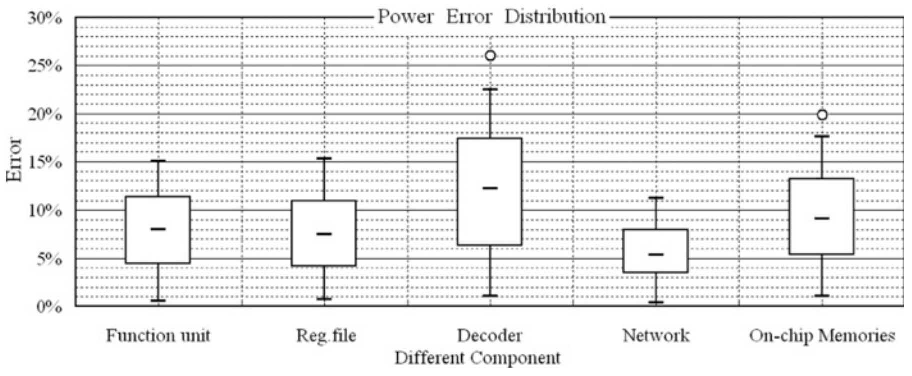

Figure 2 *Distribution of power estimation errors for different component*

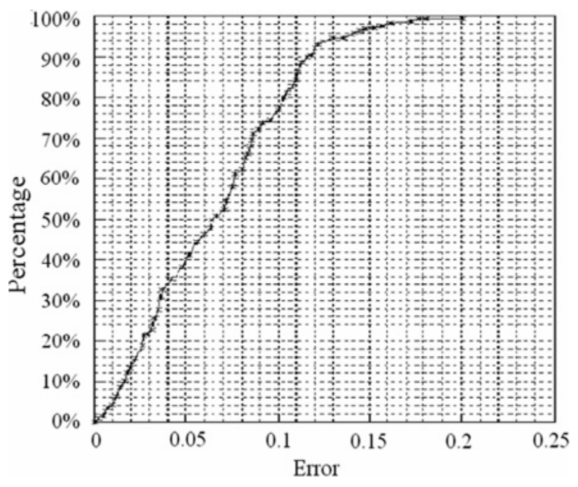

Figure 3 *Cumulative distribution of total power error*

optimisation, software pipelining, branch prediction, speculative scheduling and so on, to improve the underlying parallelism. At last, the parallel simulator with proposed cost model is used to verify the parallel codes and collect the cost information. We modify the machine description files of the initial core according to simulation reports, even adding some special function units to ensure the exact results.

In the second phase, the exploration process employs a combination of local optimisation and global optimisation. The local optimisation uses a heuristic-based algorithm to search for the Pareto-optimal core candidates with respect to performance energy in different area intervals, and gets rid of the redundant network connectivity. In global optimisation step, the analytical method with trace-driven simulation is employed to estimate various combinations of Pareto-optimal cores and on-chip memories. The output is the processor configuration that is optimal with respect to performance energy under different area constraints. Finally, the explored architecture is translated into Verilog descriptions, which are followed by Electronic Design Automation (EDA) design process to complete its very large scale integration (VLSI) implementation.

4.2 Computing cores exploration

The automated exploration begins with the characteristic analysis of target multimedia programs. By the front-end GCC compiler, the programs are translated into sequential codes using the translation template and the operation descriptions. During the sequential simulation on these codes, the relative code profiling files could be exported to determine the operation types, unit number and buses number. At first, the type and number of operations determine the function unit configuration. According to the profiling files, the operation type decides which units to be implemented, and then the number of units is decided according to the operation proportion. In general, the schedule details of back-end compiler should also be taken into account for a more accurate determination. However, because of the lack of initial description, the parallel schedules are not allowed and the measure of determining the unit number by operation proportion is taken temporarily. Secondly, the average active register roughly determines how many registers should be implemented. We take the pre-schedule measure to analyse the active register number while treating the functions or procedures as basic blocks after code expansion. Because the active register analysis by sequential codes always results in lots of pseudo-dependencies which influence the accuracy of register number, the relative check with the assistance of profiling files is especially performed to prevent them. The average number of active registers shows how many active registers should exist both to save the hardware cost and to retain the performance. Thirdly, the parallelism upper bound is determined through the trace analysis by the architecture-independent simulation, and it means how many buses should be used. In the work, unlimited hardware resources are hypothesised and none optimisation measure is taken.

Given the above initial configuration, the computing cores exploration is a critical step in automated approach for the highly efficient processor. To shorten the period, we adopt a divide-and-conquer method for processor optimisation and gain some performance-energy Pareto-optimal cores in different area intervals. The exploration process is described as follows.

Several extension phases are employed to decide the maximum configuration by considering the back-end schedule effect. Based on the initial computing core, the

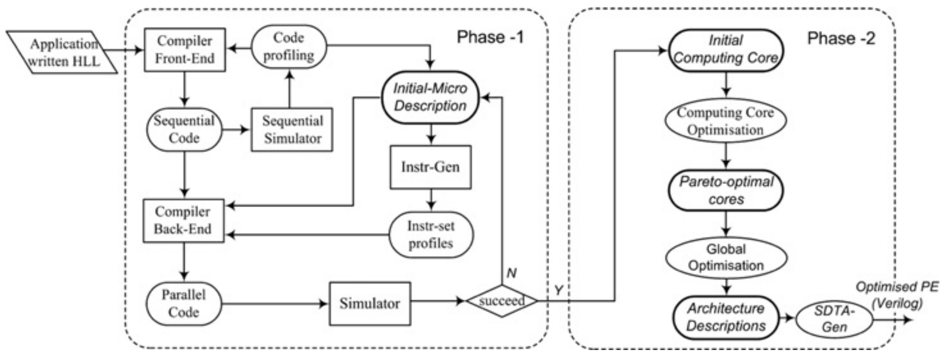

Figure 4 *Automated optimisation methodology for processor element*

back-end compiler and parallel simulator are started to collect runtime information. By marking the function units or register files above the particular utility ratios to be extendable, the number of some units needs to be increased to meet actual requirements. Each extension phase tries to increase the number of units with the highest type-utility ratio, which indicates the minimal individual utility ratio among the units with same type. The extended core with the average speedup larger than threshold α is acceptable, otherwise the selected unit type will be marked to be non-extendable. The maximum core $\mathrm{Conf}_{\max}$ is produced after several extension phases till all the unit types are marked to be non-extendable. By the way, there also exist some units which do not participate in the optimisation process, including controller unit, channel unit and so on. During the exploration, the numbers of them are set to be one.

The heuristic-based method is used to search for optimal computing cores in different area intervals. Given the maximum core $\mathrm{Conf}_{\max}$, the performance-area-energy metric of each computing core is evaluated by the cost function in (10), where Delay, A, E and Delay_0, A_0, E_0 represent the delay, area and energy costs of the current and maximum core, respectively. In general, the computing cores with higher performance but less area and energy consumption are favoured by designers. During the exploration, we generate the Pareto-optimal cores from the maximum one. Along with the area decrease during the tailor process, the increased program delay and energy consumption always result in a non-monotonous reduction of the cost function value. Therefore in order to obtain the computing cores with lower function values, those units consuming much area and energy but corresponding to small utility ratio should be cut down firstly from original cores. Heuristic function is shown in (11), and the heuristic-based algorithm for the performance-energy Pareto-optimal cores is described in Fig. 5. In the equations below, u_i denotes the utility ratio of a particular unit, A_i denotes its area parameter and E_i presents its energy. In our heuristic-based algorithm, step 1 uses the parallel simulation to obtain the delay, area and energy information of the maximum core. Then, steps 2–5 use the heuristic function to start the tailor process. While $tag[type[p_i]]$ equalling 1 means that the type of units p_i has been visited, each of iteration selects a tailored core with lower function value to be the Pareto-optimal core until the algorithm terminates when reaching the minimum one.

$$\mathrm{Quality} = \left(\frac{A}{A_0}\right)^p \left(\frac{\mathrm{Delay}}{\mathrm{Delay}_0}\right)^q \left(\frac{E}{E_0}\right)^r \qquad (10)$$

$$S = \frac{u_i}{(A_i^p \times E_i^r)} \qquad (11)$$

In addition, because of the frequent data transfers and high wire load, the transfer buses always become the critical paths in VLSI design. For these Pareto-optimal cores, connectivity optimisation transforms the fully connected configuration into a partially connected one by removing the redundant sockets to accelerate the programs and save energy consumption. To avoid the performance loss, the conservative method is adopted, where the sockets with small utility ratios and the buses without any connected socket are removed.

4.3 Processor element optimisation

The parameterise components in the design space include function units, register files, network connectivity, instruction caches and so on. The simulation-based exhaust search scheme consumes so much time that it becomes impractical, thus our optimisation process employs a

```
Step.1 Set $Conf = Conf_{max}$, $Quality_C = 1$. Using the cycle-accurate simulator to collect the
       subcomponent activities information and calculate the parameters $Dealy_0$, $A_0$, $E_0$.
Step.2 Set $Quality_{min} = \infty$, $Conf_{min} = NULL$. Assuming $n$ units of core $Conf$, we sort them
       by the ascending sequence of heuristic function values, denoted as $S_{p_1} < S_{p_2} \cdots < S_{p_n}$.
       Meanwhile, set $tag[type[p_i]]=0$;
Step.3 From $p_1$ to $p_n$, the steps $a) \sim d)$ are performed:
       a) If $tag[type[p_i]]$ equals $1$, then go back to step 3, otherwise set $tag[type[p_i]]=1$;
       b) With the cycle-accurate simulation on core $Conf - \{ fu_{p_i} \}$, the cost function value
          $Quality_{p_i}$ is derived.
       c) If $Quality_{min} > Quality_{p_i}$, then set $Quality_{min} = Quality_{p_i}$, $Conf_{min} = Conf - \{ fu_{p_i} \}$;
       d) If $Quality_C > Quality_{p_i}$, then go to **Step 4**.
Step.4 Set $Quality_C = Quality_{min}$, $Conf = Conf_{min}$, and generate Pareto-optimal computing
       core.
Step.5 If the $Conf$ is the minimum computing core, the algorithm terminates. Otherwise
       collect the subcomponent activity information from the Step 3(b) and go to **Step 2**.
```

Figure 5 *Heuristic-based algorithm for Pareto-optimal cores*

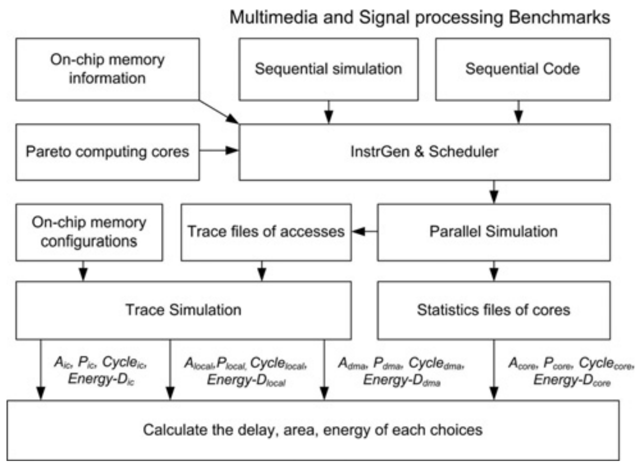
Multimedia and Signal processing Benchmarks
On-chip memory information
Sequential simulation
Sequential Code
Pareto computing cores
InstrGen & Scheduler
On-chip memory configurations
Trace files of accesses
Parallel Simulation
Trace Simulation
Statistics files of cores
A_ic, P_ic, Cycle_ic, Energy-D_ic
A_local, P_local, Cycle_local, Energy-D_local
A_dma, P_dma, Cycle_dma, Energy-D_dma
A_core, P_core, Cycle_core, Energy-D_core
Calculate the delay, area, energy of each choices

Figure 6 *Flow to collect the program statistics*

hierarchical approach. Following with the above local optimisation, the global optimisation step constructs the cross-product of Pareto-optimal cores and on-chip memories. Overall optimisation operates as follows. For the target programs, we start with the application characteristics analysis to determine the initial computing cores. Then, using the area and energy information of each component from pre-built cost library, we continually find performance-energy Pareto-optimal computing cores from entire design space, and optimise the transfer network for low power and high frequency as described in Section 4.2. The combinations of Pareto-optimal computing cores, instruction caches and local storages are used to compose different SDTA processor elements. For each computing core choice, the back-end compiler schedules the applications into target parallel codes, which are independent with the codes of other configurations. When the parallel simulation is performed on the individual parallel code, the statistical parameters about the computing components or basic units is substituted to the cost model to obtain metrics A_{core}, $\text{Cycle}_{\text{core}}$, Energy-$D_{\text{core}}$, $P_{\text{core,s}}$. With the memory trace files generated from parallel simulation, the trace-driven simulations are used to collect statistical parameters about memory accesses, which are substituted to the memory cost model to gain the parameters of instruction cache, local storage and DMA engine. Apart from the dynamic energy consumption and leakage current power of each component, such as Energy-D_{ic}, Energy-D_{local}, Energy-D_{dma} and P_{ic}, P_{local}, P_{dma}, the metrics Delay_{ic}, $\text{Delay}_{\text{local}}$ and $\text{Delay}_{\text{channel}}$ indicate the instruction cache access delays, local storage access delays and channel access delays in cycles.

We adopt the analytical performance and cost estimation approach. The area of processor element is the sum of all subcomponent areas. The runtime delay and energy consumption are analytical as in (12) and (13), where, Frequency denotes the frequency of the combined processor element. Finally, within a certain silicon area constraint, the element with the excellent characteristics of high performance and low power is selected to be the optimal one

$$\text{Delay}_{\text{PE}} = \text{Delay}_{\text{core}} + \text{Delay}_{\text{ic}} + \text{Delay}_{\text{local}} + \text{Delay}_{\text{channel}} \tag{12}$$

$$\begin{aligned}\text{Energy}_{\text{PE}} = {} & \text{Energy} - D_{\text{core}} + \text{Energy} - D_{\text{ic}} \\ & + \text{Energy} - D_{\text{local}} + \text{Energy} - D_{\text{dma}} \\ & + (P_{\text{core}} + P_{\text{dma}} + P_{\text{ic}} + P_{\text{local}}) \\ & \times \text{Delay}_{\text{PE}} \times \text{Frequency}\end{aligned} \tag{13}$$

5 Exploration experiments

According to our automated approach, the toolkits including sequential simulator, instruction generator, architecture-sensitive compiler, parallel simulator, design explorer and hardware generator have been completed. This section describes the automated process of processor element for the target domains and presents the comparison results with other methods.

The automated design of processor element is driven by target benchmarks. By the analysis to multimedia or digital signal processing applications, many representative computing-intensive kernels from TMS320C64 DSP library [16], such as FFT etc., are selected as benchmark application-specific software. These kernels are frequently used during the relative domains and their performance improvement is a key factor for the applications. In addition, the benchmarks also include some programs from MediaBench suite [17], which is considered to be a de-facto standard for multimedia domains. The speedup and low-power characteristic of these programs will prove the high efficiency of our processor element. The design space used for the exploration is listed in Table 1, and there are nearly 486 000 design choices.

Table 1 Design space of explored processor element

Components	Ranges
arithmetic	2, 3, 4, 5, 6
multiplier	1, 2, 3
logic	1, 2, 3, 4
shifter	1, 2, 3
compare	1, 2, 3
load/store	1, 2, 3
register files	6–10 banks, 8 registers per bank
buses	6, 7, 8, 9, 10
instruction cache	8, 16, 32, 64 kB
local storages	2 banks, 64 kB; 4 banks, 64 kB; 6 banks, 64 kB

5.1 Optimisation effect of computing cores

The optimisation process begins with the application analysis thereby determining the initial SDTA computing core among a huge number of choices. Based on the sequential simulation analysis towards the above baseline programs, the operation types and the operation proportion are shown in Fig. 7. The proper proportion among the arithmetic, the multiplier, the load/store, the logic, the shift, the compare and the float operations is nearly 4:1:3:1:2:1:1, which is used to decide the function unit number of individual type. Note that, during the pre-design phase, many DSP-oriented instructions [26] such as multiply accumulate (MAC) and SIMD ones have also been introduced into the elementary units to meet the parallel computation requirements. Meanwhile, according to trace analysis, the register file size and the buses number are set to be 57 and 10, respectively.

After the initialised computing core towards target domains, the threshold α in each extension phase is set to 1.002 to increase the unit number of each type. Based on $\text{Conf}_{\max}$, the heuristic function directs the units with low value to be removed. The change of constant $\langle p, q, r\rangle$ in (10) always leads to different optimisation results. In this section, the constant value $\langle 1, 2, 1\rangle$ emphasises the program performance and tends to achieve a trade-off among performance, silicon area and energy consumption. For baseline programs, the heuristic-based algorithm works as shown in Fig. 8. Given configuration $\text{Conf}_{\max}$, it spends 12 h in exploring 24 cores, and each iteration produces a Pareto-optimal one denoted by square point with respect to performance area energy. The exhaustive technique that sweeps design parameter values to consider all the design points is impractical. Instead, a local simulated annealing method is employed to validate the Pareto-optimal points. Starting from a particular optimised core, the successive disturbances by both adding high-utilised unit and removing low-utilised one become more and more unlikely with advancing search time. For each optimised core, the triangle point in Fig. 8 represents the disturbance result. It is obvious that the optimised cores by our method is a Pareto-optimal design point for which there exist no other points that achieve a better score for both area and ED^2P (energy delay delay product) metrics. Then, the connectivity optimisations are performed on these Pareto-optimal cores. The optimisation results of the six cases are shown in Fig. 9, where the average benchmark delay only increases about 3.5% and the variation of speedup is calculated to be nearly 0.001, which argues that the delays of optimised cores have not deviated from the original performance too much. On the other hand, it can be seen that the critical path is shorten with nearly 0.23 ns, and thus the program runtime has been decreased by nearly 7.6%.

5.2 Comparison with other methods

Towards the combinations of connectivity-optimised Pareto-optimal cores, instruction caches and local storages, the overall optimisation in Section 4.3 selects the designs with minimum ED^2P to be results with a particular area constraint. For the cost estimation, little overhead beyond the area constraint (e.g. 5%) is allowed by considering the error factor of our model, but the SDTA processor element with less area is preferred when the approximate ED^2P values of multiple choices are observed. With the proposed automated exploration approach, it is impractical to validate the global optimum by the simulation-based exhaustive approach. In this section, we have applied both the pruning method and genetic algorithm, thereby making an overall comparison and validating the accuracy of our results.

With the space pruning method, a probing idea is firstly employed, where we vary the amount of a single FU type and retain the configuration of other FUs so that it has the least possible influence on EP^2D values. Corner cases of the space are simulated to determine lower bounds on each unit type, and then the explored design space is reduced. The pruning result is that 1728 different cores and nine on-chip memory configurations remain. With the sequential code sampling method, it spends nearly 9 days exploring the pruned design space and gaining the optimised choices as one reference. A second approach is the genetic search which uses $1/\text{ED}^2\text{P}$ metric to calculate the fitness value. If the evaluated area of a particular design does not exceed the upper limit by a particular percentage, its $1/\text{ED}^2\text{P}$ is served as fitness value. Otherwise, the fitness value is set to a low constant. The algorithm performs the

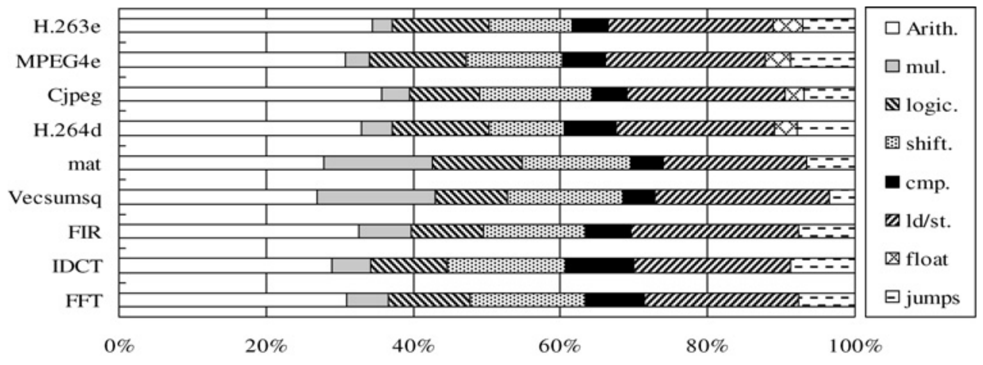

Figure 7 *Proportion percentages of baseline applications*

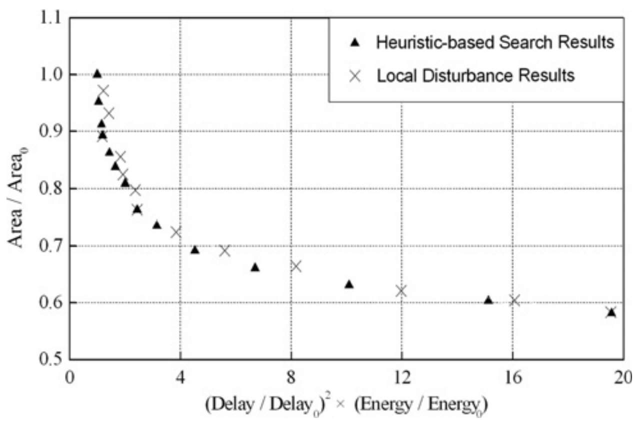

Figure 8 *Results of the heuristic-based algorithm*

evolution of individual point via processes of selection, mutation and reproduction. In practice, the number of generations during the evolution may be limited by long simulation time. For the convergence criterion, we stop the evolution when there is no longer any appreciable improvement or the algorithm runs for a fixed number of generations. In the end, the individual design with biggest fitness value is selected to be results as another reference.

A set of optimal designs are generated for target domains via the above three methods. Our proposed automated method arrives at the same optimised designs as the pruning method except for the area constraint of 6 mm^2, where our method generates the designs with 16 kB instruction cache and a big core, in contrast with the configuration of 32 kB cache and median core derived from pruning method. Fig. 10*a* illustrates the running effects of H.263 encoder program on optimised deigns derived from different methods. Firstly, the ED^{2}P value decreases with the increasing of area constraints but this trend tends to be saturated when the area constraint exceeds 7 mm^2. It indicates that the performance improvement is insignificant regardless of the extra resources at this moment. Secondly, through the comparisons on different methods, the approximate ED^{2}P values are observed because the optimised designs identified by our method are close to those by the pruning method. However, there exists the absolute difference between genetic search and our method, especially for greater area constraints. As all known, the genetic algorithm needs to evaluate many individuals during each generation thereby consuming too much simulation runtime. By considering the efficiency factor, we apply the number of maximum iterations to force its convergence, and thus the evolution process may be insufficient in some cases, where the ED^{2}P values of generated deigns by our method outperform the ones derived from genetic algorithm. In addition, Fig. 10*a* shows that the ED^{2}P values of our method are slightly larger than those of pruning methods. In general, these differences of products mainly come from delay dispersions, which are due to connectivity optimisations. All the experiments above assume that processor element frequencies have been steadied at 500 MHz, but the logic synthesis results towards the optimised designs show that the timing constraint of 2 ns cannot be satisfied in some cases. Furthermore, the frequency factor is considered in different optimised designs as shown in Fig. 10*b*, where the optimised designs using the automated approach outperform the ones by pruning method when area constraints exceed 6 mm^2.

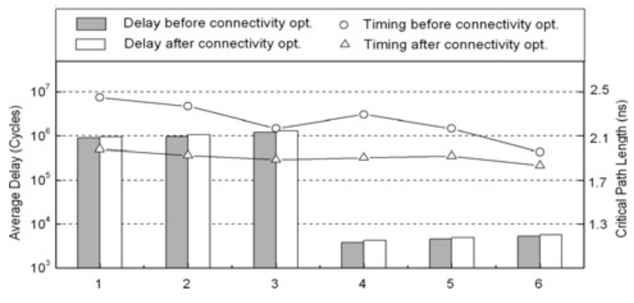

Figure 9 *Effects of the connectivity optimisations*

5.3 Efficiency evaluation

The proposed optimisation method adopts the heuristic-based exploration and the analytical overall optimisation to improve the efficiency, which is one of the most critical factors to evaluate exploration methods. To confirm it, the design time of three methods is listed below as shown in (13)–(15), respectively. Let T_{sim} be the average time per instruction for the cycle-accurate simulation, N_{instr} the number of instructions within the target programs, T_{schedule} the time for architecture-sensitive schedule on sequential codes, K_i the number of each computing unit type and M_{icache} and M_{local} the number of caches and local storages. If the simulation-based exhaustive approach is employed, the total design time is given in (13). In contrast, (14) shows the optimisation time with our automated method. T_{sequence} is the runtime for sequential simulation, N_{type} is the extension times, N_A is the number of explored cores in heuristic-based algorithm, $N_{\text{optimised}}$ is the number of Pareto-optimal cores and T_{trace} denotes the runtime for each trace-driven simulation

$$T_0 = \prod K_i \times (T_{\text{schedule}} + T_{\text{sim}} \times N_{\text{instr}} + M_{\text{local}} \times M_{\text{icache}} \times T_{\text{trace}}) \tag{13}$$

$$T1 = T_{\text{sequence}} + (N_{\text{type}} + N_A)(T_{\text{schedule}} + T_{\text{sim}} \times N_{\text{instr}}) + N_{\text{optimised}} \times M_{\text{local}} \times M_{\text{icache}} \times T_{\text{trace}} \tag{14}$$

In this work, the proposed optimisation method is performed on the Intel Pentium 2.4 GHz + Linux Red Hat 9. For the target programs, the parallel schedule takes nearly 140 s, the cycle-based simulation takes about 31 min and the trace-driven simulation for about 6 M instructions consumes about 3 min. During the experiments, the metrics N_{type}, N_A, $N_{\text{optimised}}$ are 6, 24, 14, respectively, and entire optimisation period is nearly 25 h. On the contrary,

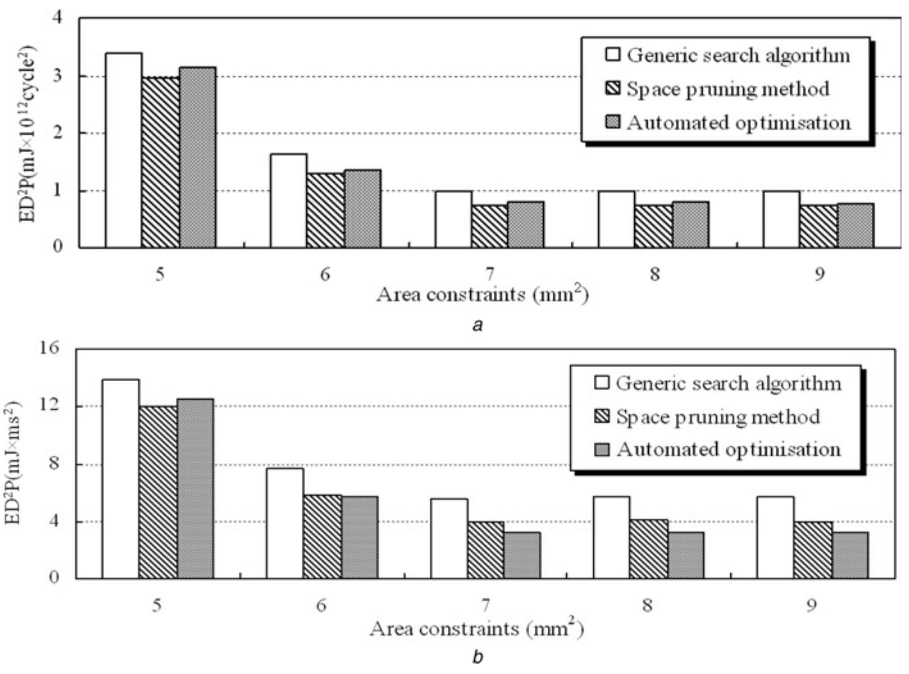

Figure 10 *Comparison of the result designs derived from different methods*
a Without any timing constraint
b With timing constraint of 2 ns

it seems to be impractical to adopt the simulation-based exhaustive method which lasts for about 30 years. Apart from these, the design time of other methods has also been evaluated for comparison. Firstly, the least time by pruning method is shown in (15), where K_i', M_{cache}' and M_{local}' represent the numbers of available components. If the pruning method is performed directly on the target programs without any code sampling method, it will take nearly 70 days. Secondly, the efficiency of genetic algorithm depends on the fitness function and population size. We spent 10 days in arriving at the relatively poor results, and the original algorithm seems to require significantly more simulations before reaching a stable solution

$$\begin{aligned} T_2 = {} & (T_{\text{schedule}} + T_{\text{sim}} \times N_{\text{instr}}) \times \prod (K_i - K_i') \\ & + T_{\text{trace}}((M_{\text{local}} - M_{\text{local}}') \\ & + (M_{\text{cache}} - M_{\text{cache}}')) + \prod K_i'(T_{\text{schedule}} \\ & + T_{\text{sim}} N_{\text{instr}} + M_{\text{local}}' M_{\text{icache}}' T_{\text{trace}}) \end{aligned} \tag{15}$$

6 Prototype implementation and performance evaluation

6.1 Processor element implementation

Using the Verilog RTL description of optimal processor element derived from the automated toolkits towards target domains, a heterogeneous multiprocessor SoC chip involving the embedded processor element is fabricated using 0.13 μm eight-metal CMOS process. The designs are optimised with the timing constraint of 2.2 ns, and their net-lists are placed and routed with Cadence SoC Encounter. As illustrated in Fig. 11, the SoC is composed of the processor element, the ARM9 processor, the memory controllers and a complete set of peripherals for the multimedia applications. Fig. 12 shows the layout of SoC with an area of 4.43 mm × 4.43 mm, where the size of processor element including 32 kB instruction cache and 64 kB local ram is nearly 7.26 mm^2 and the ARM9 processor with 8 kB instruction cache and 8 kB data cache is about 3.61 mm^2. The timing and power analysis show that our processor element can operate at 450 MHz stably while consuming about 181 mW power consumption.

Then, we break the total energy consumption of processor element down and measure the energy consumption of each component by PrimePower when running MPEG2 decoder program. The clock system consumes about 22% of total energy, while the instruction

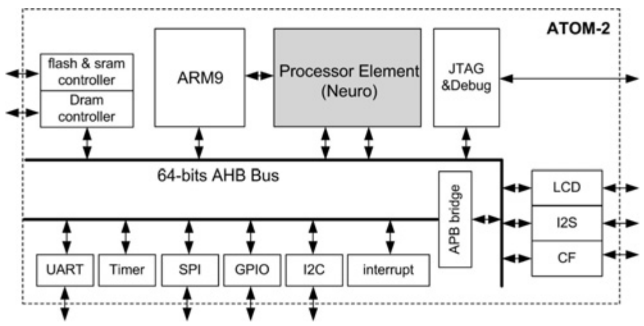

Figure 11 *Architecture of the ATOM-2 SoC*

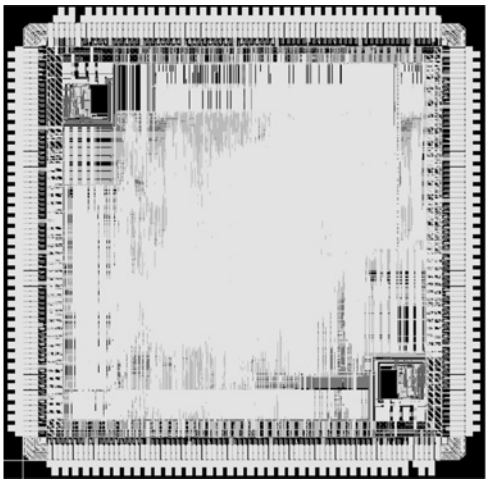

Figure 12 *Die photo of the SoC*

fetching, instruction decoding, transfer buses, functional units, register files and memory subsystem are responsible for 5.1, 9.7, 8.1, 27.4, 5 and 22.5% energy cost, respectively. Here, Yu Hu *et al.* [27] once collected the power dissipation of a VLIW processor using Trimaran simulator which had the similar configuration with us. Its experimental results revealed that the energy consumption of instruction decode unit occupied nearly 50% and even reached five times more than those of functional units. Let us assume that the above two processors consume similar energy consumption of functional units when running the same programs, it can be concluded that the decoding energy consumption of our processor element just occupies one-fourteenth of the VLIW decoding energy which plays an important role of the total energy. As all known, the TMS320C64 as one of the fastest DSPs that follow VLIW structure, contributes lots of transistor resources to the complex decode logic and thus consumes much decoding switch energy. But our processor element only corresponds to the single instruction template and in this way its simplified decoding logic brings us the advantage of low power consumption.

6.2 Performance evaluation

This section presents the performance evaluation results. The ARM9 series and TMS320C64 series processors are chosen for comparison. TMS320C64 is a VLIW architecture that is based on two identical fixed-point data paths. Each data path contains 32-bit 16-port registers and four execution units: an ALU, a shifter, a multiplier and a load/store unit. These execution units are capable of executing eight instructions in parallel. The optimised processor element includes four arithmetic units, two multiplier units, two logic units, three shift units, one compare unit, three load/store units, eight-banks register file, one channel unit, one cordic unit and so on. These units are connected by ten buses, which support the maximum ten transport instructions in parallel. The C64 series developed by Texas Instrument targets the high-performance applications, which are evaluated using the CCS3.0 [28] for C6000. Because some kernels are prepared in the TMS320C64 series DSP library, we have manually optimised the benchmarks in assembly to evaluate the performance of our optimal processor, with the assistance of the cycle-based simulator. For the fair comparison, we choose runtime cycles as the only metric, assuming the same temperature and design-for-testability technique in this section. With an optimal access timing (one cycle for each access), Table 2 presents the performance of computing cores, the performance of optimised processor outperforms TMS320C64 with an average speedup of 1.11.

In the early stage of our computing core, we have also followed the resource configuration of TMS320C64 without any design exploration, but its performance was relatively poor, only obtaining the speedup of 0.94 when running same applications. The SDTA in this paper has the advantages of exploiting multiple-level parallelisms, but copying the resources through our experience sometimes

Table 2 Performance comparisons of computing cores (assuming single-cycle access, cycles)

Application	FFT	Fir	IDCT	vecsumsq	mat	Note
ARM9	238 426	24 668	3369	906	6247	5-stage pipeline
TI-C64	9911	1048	154	46	283	8-issue VLIW, 2 data path, 2 ALU, 2 shift, 2 multiplier and 2 load/store, 32 × 2 register
optimal PE	9006	924	136	41	266	10-buses, 4 arith.units, 2 multiplier, 2 logic, 3 shifter, 1 compare, 3 load/store, 8 × 8 registers
speedup over ARM9	26.5	26.7	24.8	22.1	23.5	

FFT: A 1024-point complex FFT, 16 bits/input; Fir: A 64-sample, 16-taps, 16 bits/input complex FIR
Vecsumsq: Sum of Squares, 128 items, 16 bits; 2D DCT: 16 × 16 two-dimensional DCT
Mat_mul: the multiply of the matrices *x* and *y*, 8 × 8 matrix

Table 3 Performance comparisons on target applications (assuming 40(22)-2 access timing, cycles)

Applications	ARM9	TMS320C64	VLIW-OP	ATOM-2	Speedup over ARM9	Speedup over TMS320C64
H.264d	12.7×10^6	4.01×10^6	4.46×10^6	2.78×10^6	4.57	1.44
Cjpeg	4.8×10^6	1.08×10^6	1.57×10^6	0.67×10^6	7.16	1.61
MPEG4e	33.8×10^6	8.97×10^6	11.98×10^6	6.27×10^6	5.39	1.43
H.263e	8.7×10^6	2.95×10^6	3.42×10^6	2.11×10^6	4.12	1.40

H264d: H264 decoder, QVGA 240 × 320; Cjpeg: JPEG encoder, QCIF 176 × 144; MPEG4e: MPEG4 encoder, QVGA 240 × 320; H.263e: H263 encoder, QCIF 176 × 144

cannot reflect its superiority, or even results in the unreasonable configurations. Here, our processor elements are flexible and configurable at system design time according to application characteristics, and the optimised computing cores by our automated approach yields the obvious speedup. Then, as shown in Table 2, we also use the ARM Develop Suite to evaluate these kernels with compiler environment, and the speedups ranging from 22.1 to 26.7 have an average of 24.7.

At last, the performance evaluation to the entire SoC is presented. We evaluate the applications on the ARM-alone system and then schedule most of the computing tasks to processor element to accelerate the critical fractions of applications. In this section, we allow the ARM9 and processor element to run different tasks asynchronously. As shown in Table 3, the accelerated ratio of ATOM-2 to ARM9 achieves about 4.12-7.16. In addition, we configure the CCS3.0 simulator with the real 40(22)-2-cycle memory access timing (40 cycles for the first access outside the contents of row buffer and 22 cycles for the first access in row buffer, two cycles for each access thereafter) to collect runtime cycles of TMS320C64. Compared to TMS320C64, the ATOM-2 SoC also yields the speedup of 1.47 obtained with the fraction acceleration, the asynchronous operation and the decoupled memory accesses. Here, the application performances are also comparable to other configurable processors, such as VLIW-OP, which is derived by the design exploration towards the same target benchmarks using EPIC Explorer platform [29] and Trimaran framework [30]. It even consumes nearly 8 days to explore 11 664 configurations and the optimised choice is comprised of four integer units, one float unit, 64 GPR registers, 32 FPR registers, 32 kB instruction cache and so on. With the same DRAM access timing, the collected cycles from Trimaran simulator show that it outperforms ARM9 because of its abundance of reasonable resources, but its result is inferior to that of our processor because of many low-efficiency hardware mechanisms and little proportion of resources contributed to computations. Furthermore, the flexible and fine-grained SDTA programming model is in favour of exploiting the parallelism. Traditional VLIW operation is translated into several data transports, which allow the transfer parallelism on instruction level as well as on data level to make full use of the computing units.

7 Conclusion

For multimedia and signal processing domain, the customised processor element can provide higher performance while consuming less silicon area and energy consumption. This paper describes a novel automated approach to guide the application-specific processor element which follows the SDTA architecture to provide high performance and cost-efficient. We develop the analytical cost model with the advantages of accuracy, high efficiency and scalability and then explore the whole design space using a hierarchical method. The core exploration process adopts a heuristic-based method to search for Pareto-optimal cores with respect to the performance energy in different area intervals. The global optimisation step uses the trace-driven simulations and the analytical method to speed up the processor element process. For the multimedia benchmarks, the proposed optimisation approach outperforms the design space pruning method and genetic algorithm. Finally, an optimal processor element is implemented using UMC 0.13 μm CMOS in an SoC system. It can operate at 450 MHz while only consuming nearly 181 mW, and its core size is only 7.26 mm^2. The experimental results show that our optimised processor element with the advantage of low power consumption outperforms TMS320C64 in performance and the SoC speedups towards ARM9 and TMS320C64 for target applications have an average of 5.31 and 1.47, respectively.

8 Acknowledgments

This work is supported by National Basic Research Program (2007CB310901), Natural Science Foundation of China (60903039, 60873015) and National 863 Program (2007AA01Z101).

9 References

[1] PERICAS M., CRISTAL A., CARZOLA F.J., *ET AL.*: 'A flexible heterogeneous multi-core architecture'. Proc. PACT, 2007, pp. 13–24

[2] SHEKHAR B.: 'Thousand core chips: a technology perspective'. Proc. DAC, 2007, pp. 746–749

[3] PATTERSON D.A., HENNESSY J.L.: 'Computer architecture: a quantitative approach' (Morgan Kaufman Publish, San Francisco, 2007, 4th edn.)

[4] SAGHIR M.A.R., EL-MAJZOUB M., AKL P.: 'Customizing the data path and ISA of soft VLIW processors'. Int. Conf. on High Performance Embedded Architectures and Compilers, 2007, pp. 276–290

[5] CORPORAAL H., JANSSEN J., ARNOLD M., *ET AL.*: 'Computation in the context of transport triggered architectures', *Int. J. Parallel Program.*, 2000, **28**, (4), pp. 401–427

[6] HEKSTRA G.J., LA HEI G.D., BINGLEY P., *ET AL.*: 'TriMedia CPU64 design space exploration'. Proc. ICCD, 1999, pp. 599–606

[7] SNIDER G.: 'Spacewalker, automated design space exploration for embedded computer systems'. Technical Report HPL-2001-220, HP Laboratories Palo Alto, September 2001

[8] FORNACIARI W., SCIUTO D., SILVANO C., *ET AL.*: 'A design framework to efficiently explore energy-delay tradeoffs'. Proc. CODES, 2001, pp. 260–265

[9] AXELSSON J.: 'Architecture synthesis and partitioning of real-time systems: a comparison of three heuristic search strategies'. Proc. CODES, 1997, pp. 161–166

[10] PALESI M., GIVARGIS T.: 'Multi-objective design space exploration using genetic algorithms'. Proc. CODES, 2002, pp. 67–72

[11] PERELMAN E., HAMERLY G., CALDER B.: 'Picking statistically valid and early simulation points'. Proc. PACT, 2003, pp. 244–255

[12] WUNDERLICH R.E., WENISCH T.F., FALSAFI B., *ET AL.*: 'SMARTS: accelerating micro-architecture simulation via rigorous statistical sampling'. Proc. ISCA, 2003, pp. 84–97

[13] GRIES M., KULKARNI C., SAUER C., *ET AL.*: 'Comparing analytical modeling with simulation for network processors: a case study'. Proc. DATE, 2003, pp. 256–261

[14] KUNZLI S., POLETTI F., BENINI L., *ET AL.*: 'Combining simulation and formal methods for system-level performance analysis'. Proc. DATE, 2006, pp. 236–241

[15] KARKHANIS T.S., SMITH J.E.: 'Automated design of application specific superscalar processor: an analytical approach'. Proc. ISCA, 2007, pp. 402–411

[16] 'TMS320C64x DSP library programmer's reference' (Texas Instruments Inc., April 2002)

[17] LEE C., POTKONJAK M., MANGIONE-SMITH W.H.: 'MediaBench: a tool for evaluating and synthesizing multimedia and communications systems'. Proc. MICRO, 1997, pp. 330–335

[18] LAPPALAINEN V., LIUHA P., HÄMÄLÄINEN T.D.: 'Current research efforts in media ISA development'. Technique Report, Nokia, 2000

[19] MISHRA P., SHRIVASTAVA A., DUTT N.: 'Architecture description language (ADL)-driven software toolkit generation for architectural exploration of programmable SOCs', *ACM Trans. Des. Autom. Electron. Syst.*, 2006, **11**, (3), pp. 626–658

[20] GAN X.B., DAI K., HUANG L.B., *ET AL.*: 'A new CORDIC algorithm and software implementation based on. Synchronized data triggering architecture'. Int. Conf. on Multimedia and Ubiquitous Engineering, 2008, pp. 83–86

[21] RILEY M.W., WARNOCK J.D., WENDEL D.F.: 'Cell broadband engine processor: design and implementation', *IBM J. Res. Dev.*, 2007, **51**, (5), pp. 545–557

[22] LAI M., JIANJUN G., YASUAI L.: 'The research of an embedded processor element for multimedia domain'. Proc. MCAM, 2007, pp. 267–276

[23] VIJAYKRISHNAN N., KANDEMIR M., IRWIN M.J., *ET AL.*: 'Energy-driven integrated hardware-software optimizations using SimplePower'. Proc. ISCA, 2000, pp. 95–106

[24] Synopsys: 'Prime power, full-chip dynamic power analysis for multimillion-gate designs'

[25] KACHIGAN S.: 'Statistical analysis' (Radius Press, New York, 1986)

[26] DASU A., PANCHANATHAN S.: 'A survey of media processing approaches', *IEEE Trans. Circuits Syst. Video Technol.*, 2002, **12**, (8), pp. 633–645

[27] HU Y., LI Q., KUO C.C.J.: 'Run-time modeling and estimation of multimedia system power consumption'. IEEE Conf. on Embedded and Real-TimeComputing Systems and Applications, 2005, pp. 353–356

[28] New Code Composer Studio Tuning Edition (V 3.0). http://www.ti.com/

[29] ASCIA G., CATANIA V., PALESI M., *ET AL.*: 'EPIC-explorer: a parameterized VLIW-based platform framework for design space exploration'. International Workshop on Embedded Systems for Real-Time Multimedia, 2003, pp. 65–72

[30] MIDDHA B., GANGWAR A., KUMAR A., *ET AL.*: 'A Trimaran based framework for exploring the design space of VLIW ASIPs with coarse grain functional units'. Int. Symp. on System Synthesis, 2002, pp. 2–7