Evaluation of Application-aware Heterogeneous Embedded Systems for Performance and Energy Consumption

Jayaprakash Pisharath Nan Jiang Alok Choudhary
Department of Electrical & Computer Engineering
Northwestern University
Evanston IL - 60208 USA
{ jay, jiangjf, choudhar } @ece.northwestern.edu

Abstract

In this work, we first present an application-initiated strategy that aims to control the energy consumption, while simultaneously enhancing the performance of a heterogeneous embedded system. We assess the benefits of using this strategy by means of a traditional evaluation framework. Even though the overall benefits and improvements are apparent, the performance-energy tradeoffs are not prominently noticeable when the traditional framework is used during evaluation. Hence, we propose a framework based on a new metric called energy-resource efficiency (ERE). ERE defines a link between the performance and energy variations in a system. This metric also serves as a guide to determine the amount of resources needed to attain the desired performance and energy behavior. Our experimental results clearly indicate that a heterogeneous system running an application-aware strategy, when correctly calibrated using ERE, leads to great performance and energy gains.

1. Introduction

Embedded computing devices are becoming a part of everyday lifestyle. Smart cards, vending machines, TV set-top boxes, personal data assistants (PDAs), mobile phones, and industrial automation equipment are just some common embedded devices. Each of these devices is a congregation of various emerging technologies. For instance, devices like set-top boxes and mobile phones are a result of the merger of technologies like broadband communications or 3G wireless networks with interactive multimedia. Such embedded devices support numerous functions like multimedia (MP3, MPEG2 media playback), wireless communication (GSM, digital radio, Bluetooth) and some mandatory functions including user interfaces and file management, to be carried out at the same time. Even further, newer standards such as MPEG4, WMV and JVT (H.26L) require these devices to have a high performance in order to handle the data flow in real time. As a result, system designers are opting to embed more than one processor or processor core into these devices in order to satisfy both multitasking and performance needs [4,5,8]. Besides performance, energy consumption has also been a crucial issue in the design of such embedded systems. Techniques to curtail the energy consumption of embedded systems are already in place[2,13,15]. Researchers have also proposed mechanisms to study performance-energy tradeoffs for various systems [3,16,17]. In reality, achieving a good tradeoff is a tedious task since it requires a thorough analysis of the system under consideration, which could be time consuming.

In this paper, we first design an application-driven scheme that aims to improve the performance and concurrently reduce the energy consumption of a system. The scheme targets a multi-resource heterogeneous embedded system consisting of low-power embedded processors that serve as computing resources and a general-purpose conventional processor that acts as the master controller. We experimentally study and characterize the behavior of our system when such an application-controlled scheme is supported. The system is analyzed first using a traditional evaluation framework with *power*, *execution time*, *energy*, and *energy-delay product* [6] as the metrics.

Our experimental results highlight the lack and hence, the need for a more detailed and illustrative analysis framework for multi-resource heterogeneous environments. Likewise, resource usage is an additional and important factor influencing the performance of a multi-resource heterogeneous system, which is not considered by the traditional evaluation framework. On the basis of these factors, we present a more relevant framework to study such heterogeneous systems. In this modified evaluation framework, the improvement in performance is judged by measuring the achieved speedups and the efficiency of parallelization (which includes resource usage). We propose a new metric named energy-resource efficiency (*ERE*) to ensure the completeness of this framework. *ERE* defines the efficiency of multi-resource usage in improving the performance and reducing the energy consumption. This measure also relates the improvements in performance with the variations in energy consumption. Thus, one can analyze and hence, utilize the *ERE* in achieving a trade-off between performance and energy consumption. *ERE* also gives an estimate of the number of resources needed to achieve the desired performance and energy consumption.

The rest of the paper is organized as follows. In Section 2, we describe the heterogeneous setup that we used for our experiments. Section 3 presents our experimental results. In this section, we first present a simple analysis of the performance and energy consumption values obtained after implementing certain application-level power and performance optimizations on our architecture. Subsequently in Section 4, we define the energy-resource efficiency along with the new analysis framework, and detail its implications on performance and energy consumption. Section 5 summarizes our results.

2. Heterogeneous platform

A heterogeneous communication system with DSPs and PowerPC forms the backbone for all our experiments. Our setup involves a board with one Motorola MPC7410 PowerPC chip [10] and a farm of twelve Motorola MSC8101 DSPs [12]. The block diagram for our setup is shown in Fig. 1. The MPC7410 PowerPC (also referred to as PPC) is used only as a master controller. The MSC8101 DSPs are the actual computing resources.

We designed a centralized power-management module to handle the idle states of our system. This centralized module executing on the PPC, manages the power modes of both PPC and DSPs. By existing as an arbiter between the hardware and software layers, this module enables an application to set the entire system to various low-power modes.

The MPC7410 PPC, offers three programmable power modes, namely *doze*, *nap* and *sleep* [10]. These modes are enabled by setting certain registers. In full-power mode, MPC7410 typically consumes 6.6 watts of power, whereas in *doze*, *nap* or *sleep* mode it consumes just 3.6, 1.35 or 1.3 watts respectively [9]. Our centralized power-management module puts the master PPC to *doze* mode whenever there is no communication with the DSPs, or if the PPC is not controlling any task. Even after assigning a task to the DSPs, the PPC remains in a low-power "receive" state waiting for communication from DSPs. That is, our module shifts the PPC to the *doze* mode and then listens for any external event (data from DSP) or interrupt to break the idle state, after which the PPC returns to full-power mode. We chose *doze* mode as the low-power mode for the master controller since this mode enables the functioning of the core units, while still offering a significant reduction in power consumption (and hence return to full-power mode is faster than it is from *nap*, *sleep* modes). The MSC8101 DSP chip supports two low-power standby modes, namely the *wait* and *stop* modes[12]. The typical power consumption of full-power, *wait* and *stop* modes are 0.5, 0.25 and 0.17 watt respectively [11]. We use *wait* state as the low-power mode for DSPs in our analysis. If the DSPs are idle at any point of time, the module running on PPC instantly transfers them to the *wait* state.

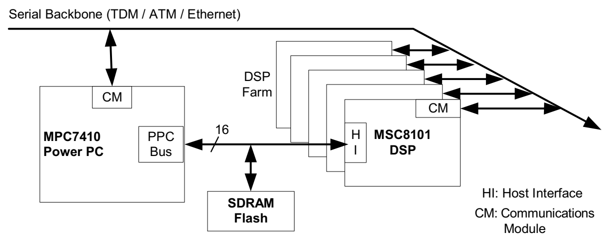

Figure 1. Heterogeneous architecture for our experiments

The PowerPC operates at 1.8V and each of the MSC8101 DSP processors at 1.5V. A HP34401A multimeter [1] connected across a small resistance (0.972 Ohms), is used for measuring the voltage, current and hence, the power consumption of our board. The HP34401A automatically adjusts the range to the characteristic being measured. The default power mode for the MPC7410 PPC, is full-power mode and for the MSC8101 DSP, it is full-power mode (of DSP). Our board boots up and stabilizes to consume 15 watts of power, which is attributed to 1 PPC, 12 DSPs, system bus and the external memory. It should be noted that this is the base power consumption of the board when none of the power optimizations are turned on. Table 1 describes our development environment.

3. Experimental evaluation

3.1. Methodology

We enhance the performance of the system by fully utilizing the available computing resources to execute an application. For this, we extend some of the existing parallelization techniques to our heterogeneous platform.

The master splits a given task among the available resources and these resources do the work in parallel. We introduce a mechanism to reduce the energy consumption by applying low-power optimizations in tandem with the parallel techniques. These low-power optimizations are implemented at the application level, and utilize the low-power modes offered by the underlying processors to reduce the overall energy consumption.

Table 1. Development environment

	PPC	DSP
Chip Name	MPC7410	MSC8101
OS	VxWorks 5.4	-
Development Environment	Tornado 2.0 (IDE)	GreenHills Multi 2000 (IDE)
Compiler	Cygnus 2.7.2 (gcc)	Optimizing C Compiler

Table 2 shows the benchmarks used for our experiments. The *art* and *bzip2* benchmarks are from the SPEC CPU2000 suite [14], whereas *g721*, *jpeg* and *pegwit* are from the MediaBench suite [7]. Each benchmark is partitioned (currently done statically, we present schemes to automate it in Section 4) in a way that allows the core segments to be run in parallel, to ensure maximum utilization of DSPs. We integrate both high-performance and power-aware techniques into the same partitioning algorithm. The following is the partitioning strategy used for implementing each benchmark in our system.

(a) After reading the input data, the master controller (PPC) splits the raw data into blocks of static size. Each block is assigned a "pending" status. The PPC assigns one of the "pending" blocks to each DSP that is participating in the execution.

(b) The DSPs then work on their respective blocks in parallel. Once the computation starts, the PPC switches itself to *doze* mode if idle.

(c) When a DSP finishes working on its block, it replies to the PPC with its output. The PPC converts the status of the received block from "pending" to "completed". If there are any more blocks with "pending" status, the returning DSP grabs another block to work on.

(d) If there are no more pending jobs to take, the returning DSP switches to *wait* mode.

(e) The PPC finally assembles the output data when all involving DSPs complete their execution.

In steps (b) and (d) of the above scheme, the low-power modes (*doze* for PPC, *wait* for DSP) are enabled by invoking the centralized power-management controller of Section 2. To recall, the controller already has schemes defined for remotely enabling low-power modes for the entire system.

3.2. Performance-energy analysis

In this section, we study each benchmark application by using a traditional evaluation framework with *execution time (delay), energy and energy-delay product (EDP)*[6] as the metrics. For each benchmark application, there is an initial part of hand-shaking between PPC and DSPs, followed by the download of code to the DSPs. This step is a prelude to the algorithm presented in Section 3.1 and the results presented in this section take these phases into account too. Moreover, the presented results are an average over many runs (typically 4) since communication is involved.

Fig. 2 to Fig. 6 shows the performance of our benchmark applications. For all applications, the power consumption increases as more DSPs are brought into the system.

For the ***art, g721*** and ***bzip2*** benchmarks, the execution time scales down linearly on designating the work to increasing number of DSPs [Fig. 2(ii), Fig. 3(ii), Fig. 4(ii)]. Furthermore, it is evident that the improvements in execution time surpass the moderate increase in power consumption. This implies that the system is very energy efficient for these algorithms. The average energy consumption decreases by 87% when 12 DSPs are used as against 1 DSP [Fig. 2(iii), Fig. 3(iii), Fig. 4(iii)].

In the case of ***jpeg***, the uncompressed data is split into blocks based on iMCU rows and the restart marker is used for removing data dependencies. Fig. 5(ii) indicates that the speedups in execution times do not scale very well with the number of DSPs (even though the execution times decrease when multiple DSPs are used). This is due to a tremendous amount of communication between the PPC and DSPs. This, in turn, has a direct impact on the energy consumption. Moreover, the overall gains in energy are much less when compared to the *art*, *g721* and *bzip2* benchmarks. The execution times have a similar effect on the energy-delay product too [Fig. 5(iv)].

Table 2. Benchmarks used in our study

Benchmark	Explanation	DSP Code Size (KB)	Input Size

art	Neural network based pattern recognition algorithm	17	10 KB image, 600 KB weight
bzip2	Data compression	19	4 MB data
g721	Voice compression	12	289 KB voice
jpeg	Image compression	10	10 MB image
pegwit	Public key encryption	29	220 KB data

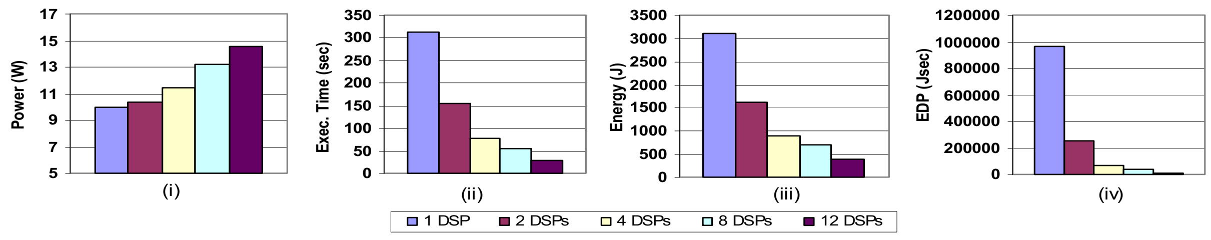

Figure 2. The power consumption (i), execution time (ii), energy (iii) and energy-delay product (iv) for *art*.

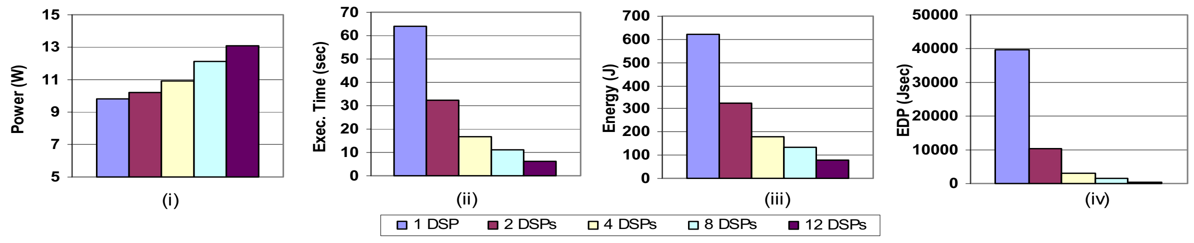

Figure 3. The power consumption (i), execution time (ii), energy (iii) and energy-delay product (iv) for *g721.*

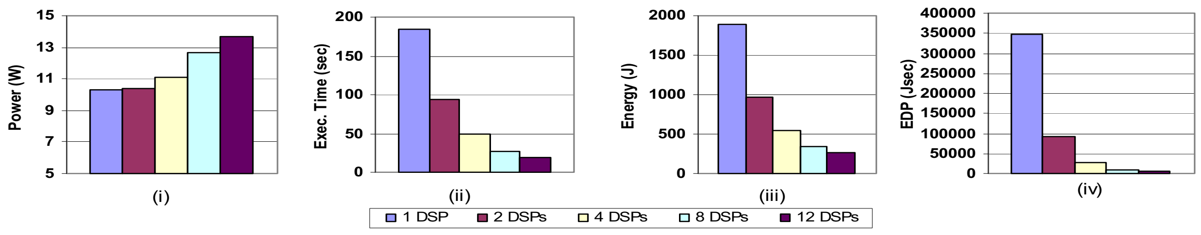

Figure 4. The power consumption (i), execution time (ii), energy (iii) and energy-delay product (iv) for *bzip2*.

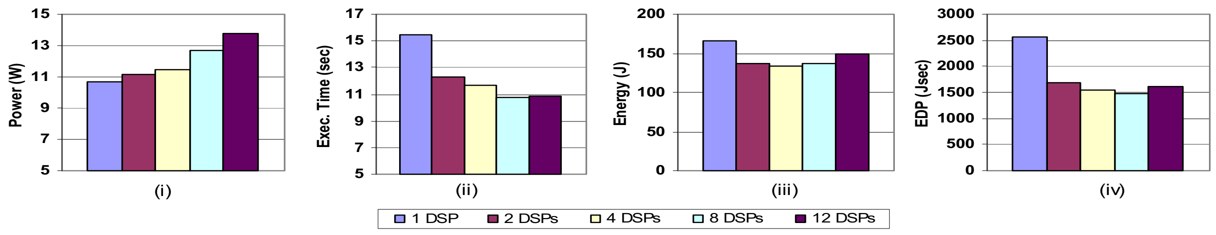

Figure 5. The power consumption (i), execution time (ii), energy (iii) and energy-delay product (iv) for *jpeg.*

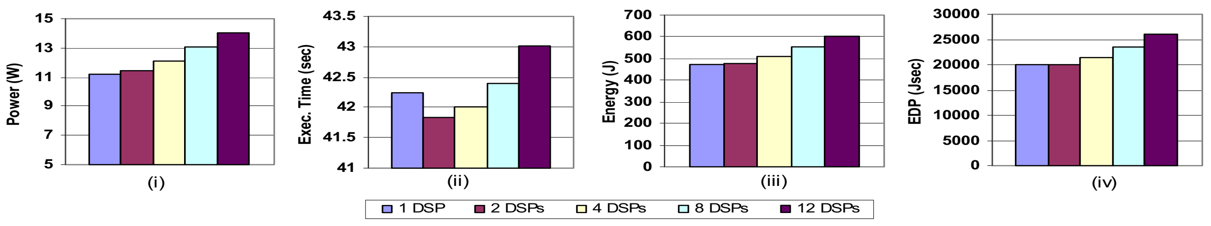

Figure 6. The power consumption (i), execution time (ii), energy (iii) and energy-delay product (iv) for *pegwit*.

pegwit is an algorithm that is very hard to parallelize and also not computationally intensive. The performance deteriorates when more than 2 DSPs are involved [Fig. 6(ii)]. This arises due to the increased communication overhead that can be avoided if fewer DSPs are used. Power consumption also increases. The worsened power and execution times consequently increase the energy consumption too. In our case, there are no improvements in execution times, energy and energy-delay product beyond 2 DSPs (Fig. 6).

Overall Observations & Shortcomings of the traditional metric:

To conclude the experimental results presented in this section, we realized that *art*, *bzip2* and *g721* are scalable in terms of both execution time and energy. That is, scalable parallel algorithms have the potential to reduce energy consumption too, besides improving performance. Additionally, for an algorithm that is hard to parallelize or that which is not scalable (like *pegwit*, *jpeg*), it is better to do the computation without much communication, i.e., with fewer DSPs. This conclusion is in accord with the observations presented by researchers in the parallel computing field. In other words, if the communication time dominates the overall execution time, the parallelization techniques prove to be useless. With respect to our case, this conclusion is applicable for both performance and energy consumption, and not just the performance.

Typically, one would be interested in studying the effect of performance improvements on the energy consumption and vice versa. Therefore, a metric that highlights performance-energy tradeoffs is essential in an analysis framework. An existing metric to study performance-energy tradeoffs is energy-delay product (EDP), which takes into account the delay and the energy consumption of a system. In our graphs, the EDP follows the trend of delay (execution time). Decreasing trends in EDP indicate good savings in both performance and energy. EDP is good at capturing overall trends. It is difficult to clearly say from EDP the mutual impact of performance and energy variations on one another. Moreover, in a multi-resource environment similar to our setup, the number of resources that are used to achieve any savings, also need to be considered during evaluation. This factor is not included in EDP or in any equivalent metric. These shortcomings and special requirements motivate us to introduce a new framework of analysis in the following section.

4. Resource-aware framework

4.1. Energy-Resource Efficiency

The primary entity that drives our framework is a new metric called Energy-Resource Efficiency (*ERE*). This metric identifies the relationship between the execution time and energy consumption. *ERE illustrates the trade-off that occurs between the energy consumed by an application and the amount of resources needed to achieve any reduction in energy consumption*. Analytically, the Energy-Resource Efficiency is defined as follows:

$$ERE = \Delta \times \eta \qquad (1)$$

where, Δ is the fraction of the energy saved by using a particular configuration, and η is the efficiency of parallelization.

$$\Delta = \frac{E_S - E_N}{E_S} \qquad (2)$$

where, E_S is the energy consumed by the non-parallelized (serial) version of the application and E_N is the energy consumed by the parallel version running on N processing elements.

$$\eta = \frac{S_p}{N} \qquad (3)$$

where, S_p is the speedup achieved through execution on N processing elements, defined as follows:

$$S_p = \frac{T_S}{T_N} \qquad (4)$$

where, T_S and T_N are the execution times of serial and parallel versions respectively.

We calculate the efficiency (η) to determine the scalability of the system for increasing number of DSP processors. The *ERE* acts as a guiding factor in identifying the amount by which energy consumption can be reduced without sacrificing much on the performance. The base case for our experiments is the configuration that has the master controller and just one DSP to do the task. The parallelized version involves more DSPs doing the same work in parallel. Table 3 shows how one can interpret the values got when using our new framework for evaluation.

Table 3. Interpretation of S_p, η, Δ, *ERE* values

Good Result	Bad Result	Impact on
$S_p > 1$	$0 < S_p < 1$	Application speedup
$\eta \bullet 1$	$\eta \bullet 0$	Efficiency of resource usage
$\Delta > 0$	$\Delta < 0$	Energy consumption
ERE > 0	*ERE* < 0	Performance-energy tradeoff

Our experimental results were analyzed using the new evaluation framework (Fig. 7). The graphs in Fig. 7 present all three aspects of what a typical system designer would be interested in, that is, efficiency of parallelization (η), fraction of savings in energy (Δ) and our newly defined *ERE*. A system designer can use one of these graphs according to one's defined needs, as shown in the succeeding paragraphs.

4.2. Trade-off Analysis using the η, Δ, *ERE* framework

Fig. 7(ii) indicates that *art* benchmark has the best efficiency closely followed by *g721* and *bzip2*. The efficiency drops tremendously for *jpeg* and *pegwit*. The drop in efficiency is attributed to the non-linear nature of speedups got when an increasing number of DSPs are involved to do the same task. The graphs further assert the results discussed in Section 3.2.

A system designer has various choices to make depending on what is desired. For instance, if the designer is looking for improvements in execution time, a quick look at the speedup of execution times [Fig. 7(i)] would be sufficient. Even further, by calculating the efficiency of parallelization (η), one can compare the improvements in speedups with the amount of parallelization. In our example, for *art*, *g721* and *bzip2*, involving 4 DSPs leads to a good speedup (~ 4) while still maintaining 100% efficiency. If efficiency is not important, we can scale up to 12 DSPs thus achieving a speedup of 11.

Now, if the system designer is interested only in improvements in energy consumption, the designer can look at the fraction of energy saved (Δ). The energy-savings (Δ) graph for all our benchmark applications is shown in Fig. 7(iii). Revisiting the discussions in Section 3.2, our goal was partly to exploit the parallel techniques to achieve reduction in energy. Power increases by a small amount when more processors are involved in processing. But the faster execution of the job, in turn, produces a positive effect on the energy. This explains the trend of the energy-savings in Fig. 7(iii). The *art*, *g721* and *bzip2* benchmarks demonstrated substantial scalability of parallelization. *jpeg* exhibits an 18% decrease in the energy consumption when parallelized to 2 DSPs, but the savings reach a saturation point beyond this level. Beyond the saturation point (around 21% energy-savings), the savings start to drop as we extend the application to more DSPs. Parallelization of *pegwit* results in a continually decreasing trend in energy-savings, thus proving its futility even in terms of energy consumption. A system designer can typically study the energy-savings graphs and deploy the corresponding number of DSPs (N) for parallelization depending on the desired savings (by defining the desired Δ).

An interesting note is that the "best" points for performance and energy-savings need not be the same. That is, a designer aiming to reduce energy savings would deploy a large number of processors (~12 DSPs) for parallelization whereas a designer looking to improve performance would look at efficiency of parallelization and would restrict to a lesser number of DSPs. Hence each graph leads to a different conclusion based on the desired trade-off. In reality, the actual cumbersome task lies in judging how much of speedup and savings in energy is desired. Ideally, we would prefer both to be high. But there usually is a trade-off between performance and energy consumption, as seen in the previous paragraphs. Additionally, using a minimal number of DSPs to achieve good performance and less energy would be best alternative. The energy-resource efficiency guides one to do all of this.

In the *ERE* graph of Fig. 7(iv), *art* and *g721* have the same performance-energy behavior. That is, their trade-off points are the same. For a designer looking at both performance and energy, the 4 processor case proves to be the best point with an *ERE* of 0.70. Beyond 4 processors, the *ERE* drops to 0.58 and then increases back (to 0.80) when 12 DSPs are used. There is only a minimal increase in *ERE* from 4 to 12 processor case, which makes the 4 DSP case a better choice. The reason is because the efficiency of parallelization dips from 100% in the 4 processor case to almost 88% in the 12 processor case. Hence it makes more sense to use lesser number of DSPs to achieve a comparable *ERE*. Looking at the *ERE* graph carefully, *art* is slightly better than *g721*, which is actually the case when comparing their energy and performance values independently too.

In the case of *bzip2*, the *ERE* values follow *g721* and *art* until the 4 DSP case. Up to this point, the algorithm utilizes the increasing number of DSPs very well to achieve improvements in both efficiency and energy-savings. Beyond this point, the *ERE* value of *bzip2* saturates to 0.69. This implies that beyond 4 processors, the improvements in performance or energy does not influence the other. Hence, depending on desired performance or energy values, a lesser or larger number of DSPs can be chosen.

In case of *pegwit*, the negative and zero values of *ERE* indicate that the parallel implementation proves to be futile towards both energy and performance improvements.

For *jpeg*, the tradeoff is clearly visible. There is definitely an improvement in both the performance and energy consumption, which is indicated by the positive values of *ERE*. The *ERE* decreases beyond 2 DSPs because the improvements in energy get negated by deteriorating efficiencies (it should be noted that there still is a speedup that is achieved). A system designer would be interested in this graph to identify the number of DSPs to deploy,

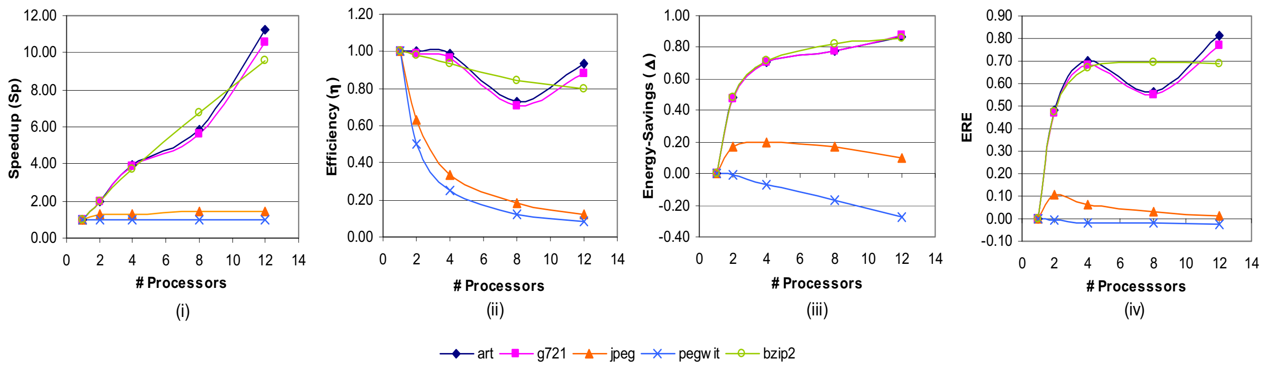

Figure 7. Speedup (i), efficiency (ii), fraction of savings in energy (iii) and energy-resource efficiency (iv) for all benchmark applications. By looking at each of the graphs, a different "best" point can be chosen. For improvements in performance, one needs to look at speedup (i) and efficiency (ii) graphs. For improvements in energy, energy-savings graph (iii) need to be chosen. For a trade-off between performance and energy, *ERE* (iv) needs to be considered.

based on the desired trade-off. For instance, in the case of *jpeg*, if only the savings in energy consumption is important, the designer can implement 4 DSPs into the system (from the Δ graph). If performance is crucial, 12 DSPs can be deployed (from S_p graph). A lesser number of DSPs should be used if efficiency of parallelization is to be considered (looking at η graph). If both performance and energy-savings are crucial, 2 DSPs would be a good number as seen from the *ERE* graph [Fig. 7(iv)]. Thus, the tradeoffs are very evident when the η, Δ and *ERE* framework is used during the analysis phase.

Overall Observations on *ERE:*

To summarize, system designers can use the energy-resource efficiency (*ERE*) metric and hence, our alternative framework to

- identify the advantages and disadvantages of a particular configuration (by looking at positive and negative *ERE* values),
- detect the breakeven point where the desired performance-energy tradeoff is achieved (by looking at the trends of *ERE* graph along with η and Δ),
- estimate the number of DSPs that are needed to achieve the desired performance-energy tradeoff (using the corresponding *ERE* values).

The advantages of using *ERE* lies in identifying energy as a design factor besides performance. By stressing on resource usage, *ERE* captures trends that cannot be captured by any equivalent metric. For instance, an equivalent metric to study performance-energy tradeoff is the energy-delay product (EDP). The *ERE* graph of *jpeg* in Fig. 7(iv) is different from its EDP graph [Fig. 7(iv)] since it considers efficiency during evaluation, which is necessary when evaluating a multi-resource environment.

The performance-energy tradeoffs are more clearly visible when the η, Δ, *ERE* framework is used during evaluation.

Our experimental results and observations also demonstrate the need for application-aware scheduling and resource allocation strategies. By incorporating some existing parallelization techniques into a power-aware scheduler, we proved that one can achieve up to 80% improvement in the performance and energy of such systems. Currently, the diagnosed results are utilized to manually partition the code by taking into consideration the desired performance-energy tradeoff. Compilers and operating systems can instead use these results in a feedback mechanism during scheduling, partitioning and resource allocation, to do a better management of the system. These techniques should adapt themselves to consider not just the performance, but the performance-energy tradeoffs, while making scheduling decisions. We suggest using a framework having speedup, efficiency, energy-savings and *ERE* instead of any traditional framework, while studying and establishing these tradeoffs.

5. Conclusions

An application-controlled strategy is very effective in reducing energy besides improving performance. We verified this by incorporating performance and power optimizations on a heterogeneous system with general purpose PPC as the master controller and a set of low-power DSPs as processing elements. There are huge performance and energy gains (up to 80%) when these application-aware techniques are applied before the scheduling stages.

From our experimental results, it is clear that energy consumption needs to be considered besides performance while assessing a system. *ERE* considers both speedup and energy variations to model performance-energy tradeoffs in a system. Performance-energy tradeoffs are prominent when the η, Δ, *ERE* framework is used during evaluation. By systematically studying a configuration using Table 3 and its relevant graphs, one can easily determine the various performance-energy tradeoffs in a system. This framework can also be used to establish a system to a set performance-energy configuration. Such features are not provided by any existing metric.

The schemes to study performance-energy tradeoffs in a system should consider the underlying environment during evaluation. For instance, our experimental base is a multi-resource heterogeneous system. The *ERE* metric identifies this fact and includes resource usage as a factor while studying the system, which is not done by any equivalent traditional metric.

Additionally, the η, Δ, *ERE* framework and the modeling technique can similarly be extended to a system having multiple processors on a single chip, like System-on-Chip setups. As a concluding note, it is evident that by using our energy-resource efficiency and a guided performance-energy tradeoff, an application-aware heterogeneous system turns out to be power efficient with substantial performance improvements.

6. Acknowledgements

This work was supported by DARPA under the contract F33615-01-C-1631. Our thanks to the other members of the Center for Parallel and Distributed Computing at Northwestern University for their help in refining this paper.

7. References

[1] 34401A Digital Multimeter Datasheet, Agilent Technologies, 2001.

[2] L. Benini and G. Micheli, System-Level Power Optimization: Techniques and Tools, *ACM Transactions on Design Automation of Electronic Systems*, ACM Press, April 2000, pp. 115-192.

[3] D. Brooks, M. Martonosi, J. Wellman, and P. Bose, Power-Performance Modeling and Tradeoff Analysis for a High End Microprocessor, *Workshop on Power-aware Computer Systems*, Cambridge, MA., November 2000.

[4] L. Hammond, B. Nayfeh, and K. Olukotun, A single-chip multiprocessor, *IEEE Computer*, IEEE Press, Vol. 30, No. 9, September 1997, pp. 79-85.

[5] Hitachi SuperH Mobile Application Processor Specifications, Hitachi Ltd., 2003. *Available HTTP:* http://global.hitachi.com/New/cnews/E/2002/0415/

[6] M. Horowitz, T. Indermaur, and R. Gonzalez, Low-power digital design, in *Proceedings of the Symposium on Low Power Electronics*, IEEE Press, 1994, pp. 8-11.

[7] C. Lee, M. Potkonjak and W.H. Mangione-Smith, MediaBench: a tool for evaluating and synthesizing multimedia and communications systems, in *Proceedings of the International Symposium on Microarchitecture*, IEEE Press, December 1997, pp. 330-335.

[8] Motorola DCT5200 Digital Set-top Terminal Specifications, Motorola Inc., 2003. *Available HTTP:* http://broadband.motorola.com/

[9] MPC7410 RISC Microprocessor Hardware Specifications, *Document No. MPC7410EC/D*, Motorola Inc., 2002.

[10] MPC7410 RISC Microprocessor Users Manual, *Document No. MPC7410 UM/D*, Motorola Inc., 2000.

[11] MSC8101 Network Digital Signal Processor, *Datasheet No. MSC8101/D*, Motorola Inc., 2002.

[12] MSC8101 16-Bit Digital Signal Processor Reference Manual, *Document No. MSC8101RM/D*, Motorola Inc., 2001.

[13] M. Pedram and J. M. Rabaey (Editors), *Power Aware Design Methodologies*, Kluwer Academic Publishers, 2002.

[14] SPEC CPU2000 V1.2, *CPU Benchmarks*, Standard Performance Evaluation Corporation, 2001.

[15] M. Tien-Chien-Lee, V. Tiwari, S. Malik, and M. Fujita, Power Analysis and Minimization Techniques for Embedded DSP Software, *IEEE Transactions on VLSI Systems*, IEEE Press, Vol. 5, No. 1, March 1997, pp. 123-125.

[16] H. Yang, R. Govindarajan, G. R. Gao, K. B, Theobald. Power-Performance Trade-offs for Energy-Efficient Architectures: A Quantitative Study, in *Proceedings of the International Conference on Computer Design*, IEEE Press, September 2002, pp. 174-179.

[17] V. Zyuban and P. Strenski, Unified methodology for resolving power-performance tradeoffs at the microarchitectural and circuit levels, in *Proceedings of the International Symposium on Low-Power Electronics and Design*, ACM Press, 2002, pp. 166-171.