Available online at www.sciencedirect.com
ScienceDirect
ELSEVIER
June 2012, 19(Suppl. 1): 79–86
www.sciencedirect.com/science/journal/10058885
The Journal of China Universities of Posts and Telecommunications
http://jcupt.xsw.bupt.cn

Optimizing energy-efficiency for program partitioning and mapping onto multi-core packet processing systems

Jing Huang[1], Olga Ormond[1], Di Ma[2], Xiaojun Wang[1] (✉)

1. School of Electronic Engineering, Dublin City University, Dublin, Ireland
2. Department of Computer Science, Iowa State University, Ames IA 50010, United States

Abstract

The sharp increase in bandwidth requirements and versatility of network applications has prompted packet processing systems to widely adopt a multi-core multi-threaded architectural design. A challenging issue when programming such a system is how to fully utilize the processing power in a pipeline-parallel topology. As the power consumption increases, maintaining the energy-efficiency of the whole system also becomes delicate.

In this paper, we proposed a strategy based on graph bi-partitioning (Bi-Par) to automatically map the programming code onto the multiple processing cores. The algorithm searches for an optimal configuration of the pipeline depth and the width of each pipeline stage. Steps taken to optimize the performance include iterations over the sub-tasks at the pipeline edges, and performing migration of tasks between cores to improve energy-efficiency. We designed a compiler framework to implement the algorithm, and use an experimental model to validate it. The simulation results show that our approach improves the energy-efficiency in all three benchmarks by between 8.04% and 34%, with a marginal loss in throughput.

Keywords packet processing systems, partitioning, program mapping, energy efficiency

1 Introduction

The main function of a packet processing system is to perform packets processing tasks at the network level. The network functionality has been greatly expanded over the past few years; and the network protocols have become much more versatile. This trend has never been stopped, or rather, is now accelerating [1]. To meet the soaring performance requirement, the multi-core platform has grown to be the de facto standard today, in terms of both the vendors' choices and researchers' focuses. The system architecture can be built upon general purpose processors such as the Intel x86-64 Xeon [2], or RISC-based network processors like Cavium's OCTEON [3] and NetLogic's XLR processors [4], or FPGA-based chips, for example, the NetFPGA project [5].

Programming in a multi-core platform however implicates several daunting issues that are not obvious or are non-existent in a single-core processor [6]. This paper looks into two of the most prominent, yet correlated, problems. The first challenge is how to schedule the miscellaneous tasks in the parallel processing cores; the second correlated challenge is how to control the overall system energy consumption under a reasonable budget. State-of-the-art network packet processing cores, such as OCTEON CN58XX, feature fast parallel processing units and hierarchical memory sub-systems. When developing applications on such a platform, either the programmer or the compiler has to know how to partition the parallel tasks and map them onto the processing cores. In theory, multi-core architectures can be configured into one of three topologies, namely pipeline, parallel or a hybrid of the two [7]. Fig. 1 illustrates a hybrid scheduling topology, where in stage 2 the cores are run in parallel and the three stages are run in pipeline connected by FIFO queues. The task mapping is flexible enough; however, how to obtain an optimal solution for a given set of applications, limited

Received date: 29-04-2012
Corresponding author: Xiaojun Wang, E-mail: xiaojun.wang@dcu.ie
DOI: 10.1016/S1005-8885(11)60464-0

processing cores and performance/latency metrics is still an open question.

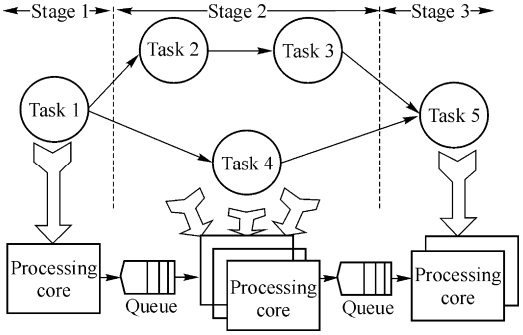

Fig. 1 Overview of multi-core packet processing system

Another prominent issue accompanying the wide adoption of multi-core systems is their greater hunger for processing power [8]. In task mapping, it is important to find a comprehensive method that takes both the system energy consumption and throughput into consideration. While it is easy to scale up the number of cores and hence the productivity, it is sometimes a self-contradictory goal to increase both the power-efficiency and the overall multi-core performance.

This paper proposed an integrated approach by extending the traditional Bi-Par [9] in program partitioning and mapping to consider the trade-off between energy consumption and system scalability and versatility. The specific contributions we make include:

1) We proposed an energy-aware method for deploying multiple network applications on a multi-core network processing system based on program partitioning and task-to-core mapping.

2) We developed a generic framework with performance and power models to evaluate the multi-core packet processing system. The system can be configured in parallel, pipeline or hybrid mode in a flexible way.

3) We gave the analysis of our approach in respect of energy-consumption and system throughput.

4) A comparison with other related work was also presented.

The focus of the paper is on our version of Bi-Par. To the best of our knowledge, this is the first work on extending Bi-Par and program mapping with energy-saving considerations. The remainder of the paper is organized as follows. Sect. 2 explains our application model and formally defines the problem we are solving. Sect. 3 describes the Bi-Par and task mapping algorithm in the multi-core packet processing system, together with a discussion of related approaches. Sect. 4 gives the results of comparison between our Bi-Par and other approaches. Finally Sect. 5 concludes our work and briefs the future research.

2 Problem statement

We use program dependence graph (PDG) as the task graph to characterize the network applications. PDG is a kind of weighted directed acyclic graph (DAG) that represents sub-function level program analysis information. The instructions of a program are grouped together to form a task by consolidating those instructions within the same basic blocks (BB). The control-flow of instructions and data-flow of variables are both categorized as dependency among the tasks. Fig. 2 shows an example PDG generated from checksum function analysis.

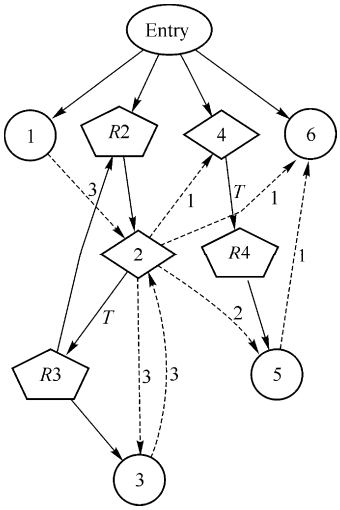

Fig. 2 A sample PDG

The round nodes contain only non-branch statements, while diamond nodes have branch instructions at the exit. Node weight is equal to the number of instructions each node contains. Pentagonal nodes are used to summarize control dependency and have zero weights. As for the edges, solid lines depict control-flow dependency and the dashed show data-flow dependency. Solid lines can be labelled as ‘true’ or ‘false’ and the dashed edges labelled with number of data transmits. The weight of the edge is equal to the communication cost to transmit the dependency.

Now we define a generic multi-core packet processing system that the application model (PDG) will be mapped onto. Let N be the number of available processing cores and each core’s instruction store size is $I\_\max$. N cores can

be configured freely in pipeline or parallel fashion like in Fig. 1. Suppose the pipeline has T stages, and in stage i the number of cores used is PE_i, then

$$\sum_{i=1}^{T} PE_i = N \tag{1}$$

In a stage, the packet latency will be determined by the sum of three factors, namely computation time, communication time between two stages, and memory access time of each stage. In this work we measure the performance from a system's viewpoint first, i.e. the system throughput.

If a task is mapped by duplication into M cores in one stage, we can take the effective computation time as a division of actual stage time by M. Multiple tasks can be mapped onto different cores in one stage, so the overall stage computation time and memory access time is subject to the slowest task. Suppose there are W tasks mapped onto one stage, then the effective stage time will be

$$\tau_{\text{stage}} = \max_{i=1}^{W}(\tau^{e}_{\text{comp}_i}) + \max_{i=1}^{W}(\tau_{\text{mem}_i}) + \tau_{\text{comm}} \tag{2}$$

where

$$\tau^{e}_{\text{comp}_i} = \frac{\tau_{\text{comp}_i}}{M} \tag{3}$$

The system throughput is decided by the slowest stage in the pipeline, so

$$\text{Thoughput} = \frac{1}{\max_{i=1}^{D}(\tau_{\text{stage}})} \tag{4}$$

and D is the pipeline length.

As for the energy consumption E, we consider the classical equation

$$E = CK_{\text{a}}V^2 \tag{5}$$

for the computational cost. K_{a} is a task-processor dependent factor and V is the voltage neither of which are considered within this paper. But the cycle runtime C is relevant here. And we measure the energy efficiency Eff as

$$\text{Eff} = \frac{\text{Throughput}}{\text{Energyconsumption}} \tag{6}$$

Instead of reducing the computational energy cost directly, we focus on improving the energy efficiency. Due to scheduling constraints (dependency) and inter-task communication delays among the cores, it is not straightforward to simply raise the ratio of packets per cycle. The energy consumption of memory interfaces and inter-stage communication should be taken into account also.

The formal definition of the problem we are solving is as follows. Given M_{app} network applications described by a PDG task graph and N processing cores that can be configured in a hybrid pipeline and parallel topology (subject to above constraints and equations), find an optimal task allocation and mapping approach that will increase the throughput rate while keeping the power consumption under control, resulting in increased energy efficiency.

3 Program Bi-Par and mapping

3.1 Base algorithm

The decision problem formulated in Sect. 2 is NP-complete [10]. To solve this we adopted a divide-and-conquer heuristic, namely program Bi-Par and recursive task mapping. The base algorithm is an application of the classical max-flow min-cut problem from network flow study [11]. The PDG is augmented as a flow network with dummy entry and exit nodes. A min-cut will partition the graph into two sub-sets where the connecting edges would incur minimum flow values. In the case of PDG, this means that the edges with lowest dependency weight between two sub-tasks will be chosen. The workflow is given in Fig. 3. A detailed explanation of each step is summarized in Table 1.

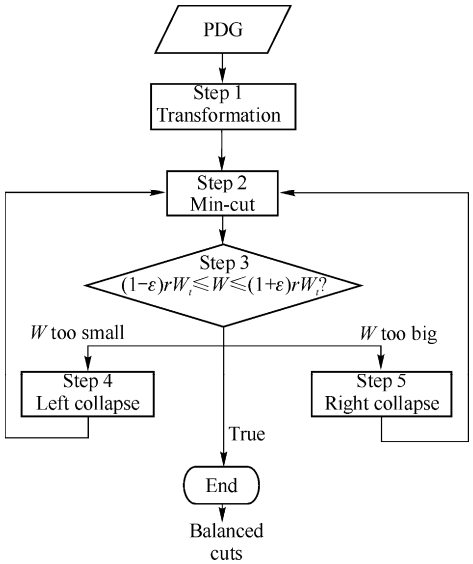

Fig. 3 Base recursive Bi-Par algorithm

Recall the equations we deduced in Sect. 2. The system throughput is determined by three factors, i.e. communication cost, computation cost and memory access time. The min-cut ensures that the algorithm always tries

to minimize the communication cost. The balanced-weight property guaranteed by the step 3 in Fig. 3 ensures that the pipeline is evenly loaded so that very little overhead would be wasted in synchronization. There is of course certain trade-offs between finding minimum communication cost and balancing the pipeline. We adopted a deviation factor ϵ to allow a flexible exploration between the two goals, as detailed in Table 1. The cutting ratio γ is measured by the weights between two cuts, and can be used to find an arbitrary number of cuts of the original program by recursively running the Bi-Par.

Table 1 Steps in recursive Bi-Par

Input	Flow graph, ε, γ
Step 1	Identify the start and terminal node
Step 2	Find a min-cut that bi-partitions the network into X and X'. Let W denotes the weights of X, and W' for X'
Step 3	If $(1-\varepsilon)\gamma W_t \leqslant W \leqslant (1+\varepsilon)\gamma W_t$, then terminate
Step 4.1	If $W < (1-\varepsilon)\gamma W_t$, then collapse all nodes in X to start node
Step 4.2	Select a node in X' and collapse it to the start node as well
Step 4.3	Go back to step 2
Step 5.1	If $W > (1+\varepsilon)\gamma W_t$, then collapse all nodes in X' to terminal node
Step 5.2	Select a node in X and collapse it to the terminal node as well
Step 5.3	Go back to step 2
Output	Two balanced cuts

After allocating the sub-tasks as indicated by the PDG cuts, we assign each task with appropriate computation resources. In our model, the nodes weight represents the computational needs (in terms of core cycles) and the edges weight labels the communication needs (interconnects between cores). So we assign each task with the number of cores in proportion to its nodes weight and the number communication interconnects in scale with the PDG edges weight.

3.2 Energy-aware extension

The algorithm described in Fig. 3 only takes throughput performance into consideration and aims solely at increasing throughput. However, as we discussed before, the energy consumption cannot be overlooked nowadays especially with the increasing number of cores on chip. So we extended the original algorithm with refinement steps using power-related data to increase the energy-efficiency. The data we profiled mainly contains:

1) The average energy consumption on each processing core. Recall that: $E = CK_aV^2$. Since V is constant here and K_a is not modifiable, we profile its number of cycles (C) for a given task together with the respective energy consumption on each core;

2) The energy consumption on interconnects. It comprises two parts, i.e. leakage energy as a function of running cycles and dynamic power related to the number of dependences between tasks on different cores;

3) Energy consumption in memory interfaces.

During the task partitioning, we collect each node's weight in terms of both execution time and total energy. In the task mapping, we iterate over the sub-tasks residing at the edges of the graph between cuts, migrate each of them to neighbouring cores and find out which migration would reduce the product of stage time (in cycles) and energy consumption (in J) the most, thus improving the energy-efficiency as given in Eq. (6) (line 4 to 11).

The intuition behind the refinement heuristic is that by migrating boundary nodes, a large search scope is available for optimizing energy-efficiency at the cost of a small throughput sacrifice. The proposed technique tries to identify any groupings of nodes with uniform memory accesses in order to minimize memory interface leakage. Interconnects leakage power is saved by turning off interconnects within un-balanced pipeline.

Energy-aware Bi-Par algorithm
Input: task graph $G(V, E, W_v, W_e)$, list of possible cores numbers
Output: task mapping matrixes;
for each number of cores N
Bi-Par (G, N)
Compute stage time and energy consumption for two cuts respectively, T_1, C_1, T_2, C_2
for each boundary nodes B_i
try migrate B_i to the neighbour cut
re-compute T_1', C_1', T_2', C_2'
if $\frac{T_1+T_2}{T_1'+T_2'} > \frac{C_1'+C_2'}{C_1+C_2}$ then
update the cut
$C_1 = C_1'$, $C_2 = C_2'$, $T_1 = T_1'$, $T_2 = T_2'$,
end if
end for
allocate cores based on cut_ratio γ
if pipeline not even or code size > limit
Bi-Par (G_i, N_i) /* recursive Bi-Par*/
same migration trials in recursive Bi-Par
end if
for the number of stages S, record the task mapping in a matrix $\boldsymbol{M}[S,N]$
end for
return ($\boldsymbol{M}_1[S_1,N_1]$, $\boldsymbol{M}_2[S_2,N_2]$…$\boldsymbol{M}_k[S_k,N_k]$)

3.3 Other approaches

A vast array of literature exists in the area of task allocation and mapping for multi-threaded and/or

multi-core system [13–16]. As the focus of our work is on network processing applications, we compare our approach mainly with the studies in the networking area.

The early work proposed by Weng [17] employed randomization in program mapping. The tasks are randomly allocated to processing cores without violating dependency constraints. All valid mappings are recorded and the one with best throughput is filtered out in the second phase of the strategy. Near-optimal mapping is not guaranteed especially when the iteration time is limited.

Another heuristic described in Ref. [18] is based on greedy algorithm. It packs the task by filling one processing core with BB until the instruction store is full. However, it does not take communication cost into consideration; so the mapping quality could be sub-optimal.

Our work resembles the approach discussed in Ref. [19] most. Yu et al also adapted Bi-Par for network processors. Their refinement focuses on throughput optimization and does not include energy awareness. In our experiments, we compared our results against these three approaches [17–19] and give a comprehensive comparison analysis.

In Ref. [20] Kuang and Bhuyan took power budget into consideration for task scheduling in packet processing system. However, their approach is based on dynamical voltage and frequency scaling (DVFS) which needs hardware support. Additionally, their method reduces power by extending the computation time, rather than optimizing energy-efficiency. In this regard, we do not compare with their approach in this paper.

4 Performance and energy-consumption evaluation

To validate our solution, we implemented a simulation framework to allow easy and large design space exploration. It has the performance and energy models respectively. In this section we will describe the experiments and discuss the results we collected using our models.

4.1 Testbench framework

We extended the SUIF/Machsuif compiler [21] with new passes that perform code analysis, PDG generation and Bi-Par mapping. Fig. 4 depicts the brief components and workflow of our test-bench. The application is first profiled with Halt passes provided by Machsuif [21] and the task graph with profiling analysis is fed into the PDG generation pass. The PDG module will collect all the information in an internal augmented PDG. Then program partitioning and mapping is carried out over the PDG. Task mapping results are input to the simulator to give performance and energy results. This process can be recursively executed to conduct comparison and optimization for a given application or a set of applications.

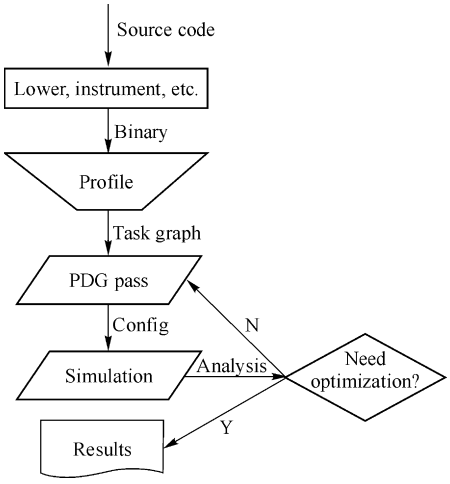

Fig. 4 Experiment framework

4.2 Performance results

In the system-level, the total throughput of a network processing system is the decisive measurement of the performance. However, the individual packet latency is also an important factor in many applications, e.g. real-time streaming. For comparison, we use the framework to evaluate three other approaches from the literature described in Sect. 3. The benchmark applications are LC-Trie IP-forwarding, IP packet encryption (IPsec) and Port-Scan adopted from PacketBench [22]. The core frequency was set to 2 GHz. Both memory access time and interconnects transmission time are assumed to be one unit of clock cycle.

Table 3 shows the throughput measurements for different combinations of the three applications with different numbers of processing cores. Since the code size limit is seldom a bottleneck for modern multi-core network systems, the number of pipeline stages in our evaluation were short. Thus the even number of cores is preferred to enable parallel processing across pipelines. In Table 2, application I is LC-Trie, II is IPSec, and III is Port-Scan. For all of the applications, our energy-aware Bi-Par exhibits good scalability as the number of cores increase.

The throughput gain is greater than double when cores are added from 8 to 16 and upwards. This is due to the free migration of tasks that have high communications cost between pipelines when processing resources are abundant. Bi-Par favours communication-heavy applications over computation-heavy ones since the base algorithm minimizes inter-stage communication cost. The throughput increase from 16 to 32 cores for PortScan is 269% while for IPsec is 244% in this case.

Table 2 Throughput for combinations of three applications in multiple cores

	I	II	III	I + II	II + III	I + III	I + II + III
2 Cores	0.56	0.18	0.11	0.11	0.05	0.07	N/A
4 Cores	0.91	0.33	0.28	0.18	0.11	0.12	0.04
8 Cores	1.65	0.75	0.6	0.41	0.39	0.41	0.12
16 Cores	3.78	1.98	1.43	1.12	0.88	0.9	0.41
32 Cores	8.75	4.85	3.85	3.1	2.12	2.43	1.45

To avoid any potential bias, we used LC-Trie plus IPsec in the performance comparison experiments. Fig. 5 illustrates the results when 16 cores are used for mapping the two applications. By varying the number of stages, we are simulating different requirement for task code sizes. Our approach (BiPar-E) shows 33.1% throughput improvement over greedy in a 2-stage pipeline and 50.7% over randomization in a 4-stage pipeline. Randomization requires very large search space as we discussed. When the pipeline is longer (i.e. more applications) and search time is predefined, it is hard to reach a good mapping. Our energy-aware extension brings an average of 10% throughput decrease compared to Bi-Par without migration. We will revisit it with energy consumption data to validate if the efficiency is improved.

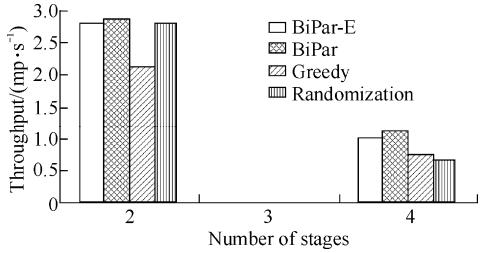

Fig. 5 Throupugh comparison by number of stages

Fig. 6 summarizes the individual packet latency comparison for the three benchmark applications. For LC-Trie, four approaches generate similar results. For the other two applications, the latency difference is within 10% margin among the four approaches. And our extension involved a slight 5% increase on average. A safe conclusion is that the energy-aware Bi-Par would not sabotage the individual packet latency even if we optimize for system throughput.

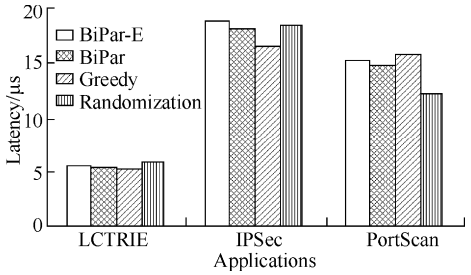

Fig. 6 Latency comparison by applications

4.3 Energy results

We measure the energy efficiency of our algorithm as the system throughput divided by total energy consumption (in J). The runtime power is usually an important indication of the energy-efficiency. However, traditional techniques such as DVFS just try to reduce power at the expense of longer runtime cycles. The total energy consumption could be well the same if not more in that case, implying that the energy-efficiency is not improved. Here we organized our energy data from an efficiency perspective as depicted in Fig. 7. The bar graph shows the total energy consumed by processing one million packets with three benchmarks respectively and in increasing order by the number of processing cores. The trend-line illustrates the energy-efficiency by graphing the throughput (in mp/s) over energy consumption (in J). In all benchmarks, the energy-efficiency is clearly on the rise as we scale up the number of cores. It proves our energy-aware Bi-Par to be particularly beneficial in a large system with dozens of processing cores. For LC-Trie, we noted 25.4% increase of energy-efficiency when cores are populated up from 2 to 16. The corresponding increase for IPsec is 168% and 29.4% for PortScan. The dramatic rise for IPsec is majorly attributed to the little heat overhead in interconnects and memory interface, especially the leakage power (which is considerably larger in LC-Trie).

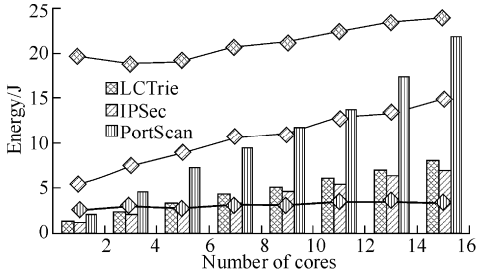

Fig. 7 Energy consumption comparison by applications

To explicitly demonstrate the energy-efficiency gains of our Bi-Par and extension, we collected the energy-related data of other three approaches in our simulator as well. We used 8 processing cores and set each core's maximum code size to be 2 000. The results are shown in Fig. 8. For all the three benchmarks, our algorithm not only excels the original Bi-Par without energy-aware refinement, but also generates better mappings than greedy and randomization. In IPsec, BiPar-E gained 34% energy-efficiency increase by migrating tasks in the refinement step. The outstanding gain is mainly because the availability of many sub-tasks at edges and little back-dependency among them. The power on processing core is the decisive factor for IPsec so the migration can take considerable effect. By nature, migration refinement can have little impact on memory and interconnects energy consumption except for leakage power. Yet in LC-Trie and PortScan we still observed an average of 10% efficiency improvement after refinement step. Therefore, our algorithm proves promising and advantageous both in terms of scalability and universality.

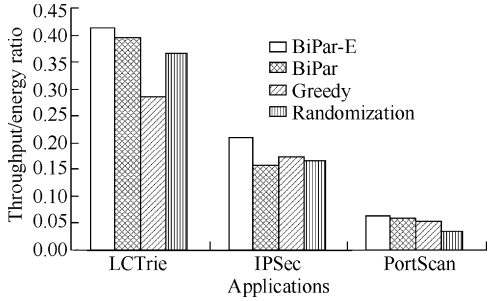

Fig. 8 Energy efficiency comparison by applications

5 Conclusions

In this work, we proposed an energy-efficient program partitioning and mapping algorithm for packet processing systems. The approach is based on Bi-Par and built into a compiler suite. We also implemented an evaluation framework to simulate the multi-core network processing system in terms of performance and energy consumption. We observe 8.04%–34% increase in energy-efficiency by applying our refinement with only a slight throughput loss in comparison with three other partitioning and mapping algorithms, i.e. greedy, randomization and base Bi-Par.

The current algorithm looks at BB of the program code for partitioning and migrations. In future work, we plan to extend our algorithm into lower level like instructions to fine-tune the energy-efficiency refinement step.

Acknowledgements

This work was supported by the Embark Initiative of the Irish Research Council for Science, Engineering and Technology and by the European FP7 ECONET Project (258454).

References

1. Bolla R, Bruschi R, Davoli F, et al. Energy efficiency in the future Internet: a survey of existing approaches and trends in energy-aware fixed network infrastructures. Communications Surveys & Tutorials, IEEE. Second Quarter, 2011, 13(2)
2. Xia G, Liu B. Accelerating network applications on X86-64 platforms. IEEE Symposium on Computers and Communications (ISCC). Jun 2010
3. Meng J, Chen X, Chen Z, et al. Towards high-performance IPsec on cavium OCTEON platform. Trusted Systems. 2011, 6802
4. Halfhill T R. Netlogic broadens XLP family. Microprocessor Rep.. 2010, 24
5. Yin D, Unnikrishnan D, Liao Y, et al. Customizing virtual networks with partial FPGA reconfiguration. ACM SIGCOMM workshop on Virtualized Infrastructure Systems and Architectures (VISA). 2010
6. Hoare C A R. Communicating sequential processes. Commun. ACM 26. Jan 1983, 1: 100–106
7. Yao J, Luo Y, Bhuyan, et al. Optimal network processor topologies for efficient packet processing. Global Telecommunications Conference, GLOBECOM. Nov 2005
8. Chang P, Wu I, Shann J J, et al. ETAHM: an energy-aware task allocation algorithm for heterogeneous multiprocessor. Design Automation Conference (DAC). Jun 2008
9. Yang H H, Wong D F. Balanced partitioning. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems. Dec 1996
10. Plishker W, Ravindran K, Shah N. Automated task allocation on single chip, hardware multithreaded, multiprocessor systems. Workshop on Embedded Parallel Architectures (WEPA). 2004
11. Yang H, Wong D F. Efficient network flow based min-cut balanced partitioning. IEEE/ACM International Conference on Computer-Aided Design. 1994
12. Mishra R, Rastogi N, Zhu D, et al. Energy aware scheduling for distributed real-time systems. Parallel and Distributed Processing Symposium. Apr 2003
13. Zhang Y, Ootsu K, Yokota T, et al. Automatic thread decomposition for pipelined multithreading. IEEE 16th International Conference on Parallel and Distributed Systems (ICPADS). Dec 2010
14. Dai J, Huang B, Li L, et al. Automatically partitioning packet processing applications for pipelined architectures. ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI). Jun 2005
15. Mallik A, Zhang Y, Memik G. Automated task distribution in multicore network processors using statistical analysis. The 3rd ACM/IEEE Symposium on Architecture for Networking and Communications Systems (ANCS). 2007
16. Li L, Huang B, Dai J, et al. Automatic multithreading and multiprocessing of C programs for IXP. The tenth ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming (PPoPP), 2005
17. Weng N, Wolf T. Profiling and mapping of parallel workloads on network processors. 20th Annual ACM Symposium on Applied Computing. Mar 2005

18. Yao J, Luo Y, Bhuyan L, et al. Optimal network processor topologies for efficient packet processing. GLOBECOM'05: IEEE Global Telecommunications Conference. Dec 2005
19. Yu J, Yao J, Bhuyan L, et al. Program mapping onto network processors by recursive bipartitioning and refining. 44th ACM/IEEE Design Automation Conference (DAC). 2007
20. Kuang J, Bhuyan L. Optimizing throughput and latency under given power budget for network packet processing. IEEE Conference on Computer Communications. 2010
21. Lam M. An overview of the SUIF2 system. ACM Conference on Programming Language Design and Implementation (SIGPLAN). 2009
22. Ramaswamy R, Wolf T. PacketBench: a tool for workload characterization of network processing. IEEE International Workshop on Workload Characterization. 2003

From p. 78

References

1. Cui T, Tellambura C. Blind receiver design for OFDM systems over doubly selective channels. IEEE Trans. Commun.. May 2007, 55(5): 906–917
2. Bingham J A C. Multicarrier modulation for data transmission: an idea whose time has come. IEEE Commun. Mag..May 1990: 5–14
3. Beek J, Edfors O, Sandell M, et al. OFDM channel estimation by singular value decomposition. IEEE Trans. Commun.. Jul 1998, 46: 931–939
4. Tanabe N, Furukawa T, Tsujii S. Robust noise suppression algorithm with Kalman filter theory white and colored disturbance. IEICE Trans. Fundamentals. Mar 2008, E91-A(3): 818–829
5. Morelli M, Mengali U. A comparison of pilot-aided channel estimation methods for OFDM systems. IEEE Trans. Signal Process.Dce 2001, 49(12): 3065–3073
6. Steele R. Mobile radio communications. New York: IEEE, 1992
7. Cui T, Tellambura C. Blind receiver design for OFDM systems over doubly selective channels. IEEE Trans. Commun.. May 2007, 55(5): 906–917
8. Garcia M J F G, Rojo A J L. Support vector machines for robust channel estimation in OFDM. IEEE Signal Processing Letters. Jul 2006, 13(7): 397–400
9. Cheney E W. Introduction to Approximation Theory. New York: Mc-Graw-Hill, 1966
10. Wang X, Liu K J R. An adaptive channel estimation algorithm using time-frequency polynomial model for OFDM. Applied Signal Process. 2002: 818–830