Journal of Cloud Computing
a SpringerOpen Journal

RESEARCH Open Access

Software level green computing for large scale systems

Faiza Fakhar[1*], Barkha Javed[1], Raihan ur Rasool[1], Owais Malik[1] and Khurram Zulfiqar[2]

* Correspondence: 10msitffakhar@seecs.edu.pk
[1]School of Electrical Engineering & Computer Science, National University of Science & Technology, Islamabad, Pakistan
Full list of author information is available at the end of the article

Abstract

Energy conservation has become a critical issue in modern system electronic devices. Energy wastage in electronic devices occurs in both hardware and software components. Software drives the hardware thus decisions taken during software design and development have significant impact on energy consumption of a computing system. Green Computing addresses energy conservation by application of different techniques at software and hardware level. Energy efficient compiler is a software level green computing technique. Besides compiler optimization, an effective scheduling approach makes efficient use of resources to directly impact the green aspect. Therefore, focus of this paper is identification of energy conservation measures for software level and there utilization at compiler and scheduler. A Distributed Green Compiler (DGC) is presented in this research that is hardware independent and uses an existing distributed compiler. It distributes source code of software over a network, reshapes binary code by applying green strategies during code transformation at compile time and gives green suggestion to software programmer for energy conservation. For scheduling, Distributed Interactive Engineering Toolbox (DIET) scheduler is used and a new algorithm is proposed for the DIET scheduler. The proposed algorithm introduces green aspect in scheduler to effectually make use of resources in such a way that consumption of power and carbon dioxide emission is reduced. Performance analysis of proposed compiler shows that it conserves energy clock cycles up to 40% by applying few green strategies.

Keywords: Distributed green compiler (DGC), Energy aware compiler (EAC), Energy conservation, DIET scheduler, Green aspects, GNU compiler collection (GCC)

Introduction

Global warming has stimulated the need to rethink environmental impact of technology. Green or environment friendly computing attempts to reduce consumption of energy to reduce consumption of fuel required to produce it that entails the toxic impact on environment. The increasing need for tighter energy budgets demands vigilant energy conservation and thus driving us to propose energy aware hardware and software [1]. Effective Energy conservation however is an accumulation of both design and best practices.

High performance and energy conservation are conflicting goals in green computing. One way to conserve energy can be to reduce logic voltages; however, this causes slower circuits and low frequencies, which leads to degradation of performance [2]. In

the past few years, cloud computing has gained much popularity as it reduces execution time of a program by distributing it on different machines over a network. To solve a large computationally intensive problem, a cluster of several low capacity machines is more beneficial in terms of cost and performance compared to a high capacity machine. Because a high capacity machine does not fully utilize its resources at a time, hence it consumes more energy. Therefore, significant performance improvement with energy conservation can be achieved through distributed or cloud computing among green strategies implementation.

Optimizations for energy conservation can be made at hardware and software levels. Hardware level energy optimizations are achieved through circuit design by implementing smaller silicon process geometries, auto idle detection circuits and active well biasing techniques [3]. Software level energy optimization are implemented in operating system through Green Scheduling techniques that analyze active processes for energy requirements, and by the *Green Compilers* through program analysis at compile-time and code reshaping during transformations. Energy can also be conserved, during software development life cycle such as software analysis and design, by applying different green aspects mentioned further in subsequent sections. Furthermore, the use of energy aware software tools can help in achieving software level energy optimization [4,5]. Figure 1 shows different layers of a possible green operating system. Some energy aware programs residing in OS kernel facilitate higher layers to achieve energy conservation.

Focus of this paper is to present research on two software level optimization for energy conservation one of them is energy aware compiler and other is energy conserve scheduler. Both proposed techniques did not have any relation between them. However, programmers can use both approaches at a time.

Compiler is software that facilitates programmers to describe the solution of their problems at an abstraction level and then translates that abstraction into machine-readable form [5]. They are good source to optimize energy on software level. Most often energy aware compilers are used by software developers to designed embedded

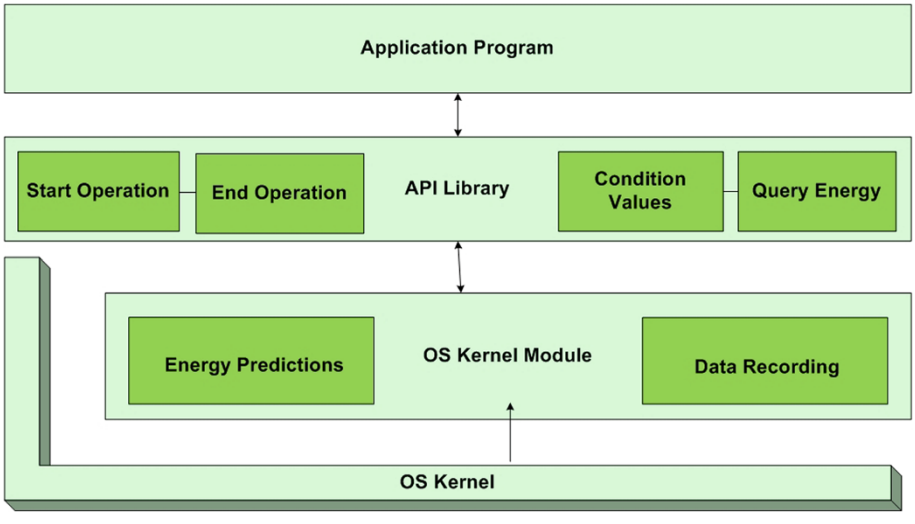

Figure 1 Energy aware computer system architecture.

systems, thus they are hardware dependent. Major contribution of this paper is to propose an energy conservative distributed compiler that uses green techniques during compilation to generate an optimized energy conservative executable. However, performance and energy conservation are conflicting goals, but the aim of this paper is tradeoff between both of them. Therefore, executable formed by green compiler will conserve energy as well as substantially maintain performance of compiler by distributing code over network at compile time.

To effectively make use of cloud resources, a good scheduler is required in cloud. Keeping this in view, DIET cloud has emerged which makes use of the cloud resources in a transparent manner. DIET architecture provides the scheduling at agents level that selects and allocates the machines based on user requirements. The use of DIET toolkit in cloud computing has given the notion of DIET cloud [6,7]. This research also presents a review of various schedulers reported in literature and scheduling in DIET is modified to introduce the green aspect in scheduling. In this paper we will be using the term power and energy interchangeably.

Rest of the paper is organized as follows: section II discusses related work of DGC and green scheduler. Several avenues for energy optimization are identified in section III. Section IV describes green strategies for software developers. Detail of proposed DGC (Distributed Green Compiler) is discussed in section V. Section VI gives details of green scheduler. Finally, Section VII concludes this research.

Related work

A. Compiler literature

ESG (*Embedded Systems Groups)* department of computer science at Dortmund University Germany has developed encc an energy aware C compiler. encc has LANCE frontend that takes C program as input. encc backend selects code using tree pattern matching algorithm and allocates registers for selected code through graph based heuristic method. After register and code selection, backend converts the intermediate code by applying optimization techniques including common sub expressions, dead code elimination, register pipelining, instruction scheduling, jump optimization and mapping of program object to different type of memory. Assembler and linker execute this code to form a binary executable. encc maintains database about energy consumption statistics of each instruction as well as memory access. Profiler take gathered information from simulator and database. These statistics are summed up and the performance statistic of complete program is generated [8].

Coffee compiler for C language is combination of software and customized hardware to achieve energy conservation at compile time [9]. A major weakness of coffee compiler and encc is hardware dependency since they are designed for embedded systems. Furthermore, green strategies implementation increases compile time especially for large projects, which causes performance degradation.

mrcc is a distributed open source C compiler using Map Reduce on Hadoop platform, but it can be used in any other cloud computing platform. It does preprocessing of an input source file, which include placement of header files on source files. mrcc checks dependency among different source files and if the source file is safe to compile on several slaves it divides them on different machine using map reduce for compilation [10]. It is a good approach towards distributed compilers but does not give energy conservative executable.

Proposed *DGC* is a hardware independent compiler that does not require any special hardware. It optimizes and reshapes source code in energy conservative executable by implementing software level green strategies. Several compilation problems are very large and cannot be compiled efficiently on a single machine. Thus a distributed environment can be a good solution for these problems, but hard for small compilation problems. *DGC* is able to perform program compilation with both approaches. A program contains several green aspects; some of them cannot be handled by energy aware compilers for example recursion elimination etc. Therefore, these aspects are skipped during energy aware executable generation process, and hence cause inefficiency. *DGC* facilitates its programmers by highlighting sensitive areas of program which consume extra energy and cannot be reshaped by compiler.

B. Scheduler literature

Grid computing provides the efficiency in terms of availability of number of distributed resources that perform the computation. In [11] a scheduling system for grid is introduced which encompasses three phases: resource discovery that lists all the available resources, system selection based on gathered information, and file staging and cleanup done by job execution. The proposed strategy has disadvantage in term of involvement of users in scheduling decision.

In [12], Chameleon scheduler is proposed that is designed for computational as well as data grid. This scheduler works well for large data applications and replicated data in grid environment. Chameleon scheduler works well not only for finding the appropriate location/machine for computation but also performs efficiently to locate the required data. In [13] a scheduler is proposed that not only dynamically allocates the resources but also define the mechanism for inter component communication.

In [14], stealth scheduler is introduced which is specifically designed for Workstation-based Distributed system (WDS). WDS refers to the distributed workstations that provide large computing capacity but WDS software system does not efficiently share the computing capacity among workstations. Moreover, the foreign processes are preempted when owner process of workstation is initiated thus causing the foreign process to wait till the end of owner process or some idle node is available. To overcome this barrier stealth scheduler does not preempt the foreign process on initiation of owner process. This avoids the unnecessary preemptive transfers thus increasing the efficiency to fully utilize the computing capacity. Two major components of stealth scheduler are StealthLS and StealthGS. StealthLS shields the effect of foreign process on owner process and it does not allow preemptive transfers thus enhancing the performance. Whereas, StealthGS allow the preemptive transfers only when required to avoid the starvation, provides transparency by automating the transfer process, and implements decentralized global scheduling thus avoiding the bottleneck that incurs in centralized environment.

Form the above description it is clear that the above mentioned schedulers lack green aspects. In [15], a green scheduler for cloud infrastructure is reported, which comprises of four algorithms. Firstly, on the basis of history prediction algorithm scheduler predicts the future request load, then on the basis of result ON/OFF algorithm turn off unused servers. Task scheduling algorithm schedules the tasks on number of machines on the basis of earliest deadline first strategy and largest capacity first strategy. Earliest

deadline first strategy queue the coming tasks whereas the largest capacity strategy is used to allocate the tasks to virtual machines. Finally, the evaluation algorithm is used to monitor the performance when the load changes.

In [16], DVFS enabled scheduling is proposed. This scheduling algorithm was originally designed for clusters then adapted in cloud with some modifications. The virtual machine request arrives at scheduler, which according to the requirement allocates the VM to the processing element (PE). PE is allocated based on voltage level. If PE with low voltage level is found and it fulfills the requirement then VM is allocated to that PE. If in case no PE is found that satisfy the requirement then PE that operates at higher voltage is selected. Moreover, with the finish in job by any VM the supply voltage o-f PE is reduced.

The effort to reduce the energy consumption is not confined to the servers only but efforts are also laid to reduce the power that is used to operate cooling systems. For this thermal aware scheduling [17] is used that schedules the job such that it reduces the overall power consumption of datacenter. Moreover, to reduce the power consumption by server, power aware scheduler [18] is designed. This scheduling approach aims at using all the processing cores in a node, which according to research, reduces the power consumption. The algorithm used in power aware scheduling is greedy-based algorithm.

It has been observed that most of the energy efficient schedulers do not take into account the network and traffic [19]. Deals with the selection of appropriate path for traffic, for this purpose tree routing topology and multi-path protocol are used. When hash function is applied by protocol then collision might occur. Because of collision two large traffic flows follows the same path leaving the other path unused. Complex central scheduler caters this problem as well as analysis of traffic in datacenter and manages the traffic accordingly. Moreover, the comparison of Simulated Annealing and Global First Fit algorithm is done and it has been shown by the experiment that simulated annealing outperforms global first fit algorithm.

With the use of virtual machines, it has been analyzed in [20] that VM live migration consumes a lot of bandwidth and number of migration can lead to the network congestion. In order to cater with this issue migration scheduler analyzes delay and bandwidth resources.

Migration scheduler also takes into account the network topology and bandwidth requirements. The schedulers proposed in [20,21] do not take into account the energy perspective in network awareness. Datacenter energy-efficient network-aware scheduling (DENS) [21] also take into account the energy efficient perspective in addition to the traffic awareness. DENS methodology balance between the network traffic, job performance, and energy consumption. Moreover, this approach is applicable to the datacenters, which performs data intensive job that requires low computation but results in heavy data/traffic generation.

Table 1 shows the summary of the schedulers. The symbol "☼" used in Table 1 shows the fulfillment of particular feature (specified in top row i.e. row 1) by the scheduler (specified in leftmost column i.e. column 1) whereas the symbol "×" shows scheduler does not cater with the particular feature. Moreover, the word "not found" in algorithms column depicts that scheduler does not explicitly states the algorithm it used.

Table 1 Summary of schedulers

Schedulers	Grid computing	Cloud computing	Green aspects	Static scheduling	Dynamic scheduling	Network awareness	Algorithm used
Grid Scheduler [11]	☼	×	×	☼	×	×	Not Found
Chameleon Scheduler [12]	☼	×	×	×	☼	×	Not Found
[13]	☼	×	×	×	×	×	Not Found
Stealth Scheduler [14]	☼	×	×	×	☼	×	Not Found
Green Scheduler [15]	×	☼	☼	×	☼	×	→History Prediction →On/Off →Task Schedule
DVFS enable Scheduling [16]	×	☼	☼	×	×	×	Not Found
Thermal-aware Scheduling [17]	×	☼	☼	×	×	×	Not Found
Power-aware Scheduling [18]	×	☼	×	×	×	×	Greedy based
Central Scheduler [19]	☼	×	×	×	☼	☼	Simulated annealing
Migration Scheduler [20]	☼	×	×	×	☼	☼	Not Found
DENS [21]	☼	×	☼	×	☼	☼	Not Found

Green strategies for compilers

Energy aware compilers analyze software programs at run time and reshape software source code by applying several green aspects during code transformation. Following are some green techniques that can be applied at local, global or inter-procedural level to make program energy aware [5].

A. Cache skipping

In programming environment loops have significant importance. In loops, replication gives high performance but causes high-energy consumption due to repetition of same thing. A good approach can be skipping of cache operations during unnecessary replication.

The study in [2] presents an efficient technique to solve cache-skipping problem by modification in compiler and hardware. In this technique compiler needs to separate the blocks that has less chance to execute for example exception block. This study presents that in ideal case there is no use of cache and hence this technique results in reduced power consumption.

B. Use of register operands

Every machine has different energy consumption cost to access resources from memory. Most studies show that memory reads and writes have higher cost as compared to use of register operands. Register operands have less abstraction than memory accesses (read/write) and therefore consume less energy.

Instructions with register operand have approximately 300 mA cost per cycle while instructions with memory operands increase this cost by 430 mA per cycle for read and 530 mA per cycle for write [2]. Thus, compilers need to use register operands

more. This will result in shorter running time and energy conservation due to exclusion of possible cache misses.

C. Instruction clustering

Some environments have special type of architecture that allows a compiler to execute pair or cluster of instructions in one cycle. For example in signal processing applications, a cluster of related or similar signals can be compiled in one run. It will reduce the running time of program and leads to energy conservation. The study in [22] shows that instruction clustering can conserve energy from 26% to 47%.

D. Instruction re-ordering and memory addressing

Sometimes the order of instructions and memory addressing is not in favour of energy safe mode. Energy consumption can be significantly reduced by changing the order of instruction to and from the power-safe mode.

A technique is proposed in [23] using *Gray Code* and *Cold Scheduling*. Gray code is used to reference consecutive memory location. The paper reports that Gray code cuts the energy consumption by 36.9% as compared to binary representation of memory. Cold scheduling algorithm for instruction scheduling uses gray code that reduces 20% to 30% instruction switching.

E. Use of energy cost database

Energy aware compilers maintain energy cost database for each transaction/instruction. This database can be used in code parsing and parse tree generation algorithms. In first run of code processing during compilation, all possible parse trees are generated and their respective energy cost is assigned using energy cost database. In the next run, less cost tree will be selected for further compilation. This mechanism ensures the selection of highly optimized energy cost tree [23].

F. Loop optimization

Several techniques are presented for loop optimization to increase energy awareness and efficiency in program. One of them is *Loop Fission*. In this technique, usually nested loops are checked across dependency graph. Dependency graph are prepared for loop body in which nodes represent statements and edge corresponds to data dependency. If there is no cycle in the graph, compiler will create loop for each statement and run them parallel using interleaved processing.

A technique in [24] uses loop optimization and memory partitioning technique to cut down energy consumption without degradation of performance.

G. Dynamic power management

Power consumption in Complementary Metal-Oxide Semiconductors (CMOS) is categorized into static and dynamic. Power consumption is dynamic when circuit is in operating state and no power leakage occurs. Whereas power consumption is static, when circuit will not be in running form but it is still powered. Dynamic power management system sets the power of its hardware in true time without degradation in performance to decrease probable power waste.

Dynamic Voltage and Frequency Scaling (DVFS), Dynamic Process and Temperature (DPT) compensation and idle time prediction are some power aware software that can drive hardware power saving mechanisms [3]. A study in [25]

present DVFS based SUIF2 compiler infrastructure for source to source level transformation. Another possibility can be brought forward if an algorithm detects its slower program regions and makes use of above techniques to cut down energy usage from the program. The results presented in [5] shows that 23% of total system energy is saved by SPECfp95 benchmark using this technique with 5% of performance degradation.

H. Resource hibernation

Hibernation is the process of using low power mode. As mentioned in Section I, idle resource can be kept in hibernate state but switching to and from this state can be wastage of precious resources and time.

A compiler algorithm reshapes a program behaviour using source level transformation in such a way that idleness threshold of a resource can be extended, and it can be switched to hibernation mode with less switching. A compiler needs to call OS directives for activeness and inactiveness of specific resource [5].

A research in [26] presents reshaping a program on a set of streamed and non-streamed applications. The study shows that energy cost can be reduced from 55% to 89% with minor performance degradation.

I. Cloud aware task mapping

Cloud aware task mapping is the use of readymade services that can be provided by different clouds. A compilation technique uses cloud services at host level for possible computation by parallel processing and keeping state records. A machine independent compiler can also use all services from remote clouds and during progression of these services; it can go in hibernation mode.

This technique is suffering from several problems; one of them is the cost of virtual machine and migration cost of host machines. Second problem is failure of network machines causing delay in compilation or other service utilization.

J. Eliminate recursion

Compiler executes recursive procedures using stack. Sometimes this technique takes a lot of space and time causing degradation of performance as well as extra energy consumption. Some compiler converts recursion into iteration. This technique may save time and energy in some cases [27].

Green strategies for software development life cycle

Energy can be conserved during software development life cycle such as software analysis, design and implementation. At design level, energy can be conserved by making energy efficient structure of software. Software implementer/developers can use following strategies during software implementation.

A. Use of green IDE& compiler

Use of energy aware or green compilers helps to conserve energy. Several open source and licensed energy aware compilers are available for example Green Hill compiler for C and C++, encc energy aware compiler for C++.

B. Use of grid & cloud computing

Grid & Cloud computing are the biggest trends that make use of readymade computer resources on demand. There are many open source software and hardware level resources available as a service by different vendors for example currency converter, calculators etc. Making use of these readymade resources in program will be beneficial in terms of energy cost and time, compare to re-programming them.

C. Recursion vs. Iteration

Recursion uses stacks. At the start of each function calls arguments have to be pushed to stack and at the end of the function call they have to be popped which takes longer execution time, and hence leads to more energy consumption. Therefore, a better approach is to use iteration and avoid recursion as much as possible during application development [22].

D. Less running time

In general, any strategy that can reduce the running time of algorithm can be helpful for reducing energy consumption [22]. Complexity of algorithm can be computed in term of Big O notations. Algorithms that have linear complexity will be more energy aware compared to those that have exponential equation of complexity.

Thus, techniques that reduce complexity of program for example avoidance of nested loops should be used during software design and development.

E. Use of energy aware data structure

Data structures have significant effect in execution of a program and energy conservation as efficient data structure is more energy conservative. Study in [22] shows that merge sort consumes less energy with array data structure whereas it consumes more energy in the case of link list data structure. Furthermore, compilation of various data structures APIs such as link list, arrays etc. with some energy conservative compiler make these data structure energy aware and use of them in program may leads energy conservation.

DGC (distributed green compiler)

This section presents the high-level workflow and basic architecture of proposed distributed green compiler.

DGC formulates an energy conservative executable by applying several green techniques to reshape source code during intermediate code conversion. A classic green compiler requires more time to compile the source code for energy conservative executable. It applies green strategies in compilation hence leads to degradation of performance. DGC decreases compilation time by distributing source code over a network of physical or virtual machines. However, software developer also has the option to compile program on a single machines.

Compiler cannot reshape all source code in energy conservative executable, for example recursion elimination, use of register operands. DGC provides green suggestions for software developers by highlighting the areas of source code, which cannot be reshaped by compiler for energy optimization during intermediate code conversion. DGC gives energy consumption statistics of program after compilation that tells programmer how much energy can be conserved in a produced executable.

Figure 2 shows the generic algorithm for DGC. A C language program will be given as input along with some optional parameters (e.g. switch, IP addresses and etc.). Switch will be used to tell the compiler that program requires distribution on network and IP addresses use to specify network machines available for compilation.

We have developed a prototype of DGC that uses distcc [28] as baseline. distcc is an open source distributed C/C++ compiler that uses GCC compiler. Distcc sends preprocessed source code across the network and volunteer machine compiles that code. Volunteer machine requires running of distcc daemon and GCC compiler [28].

Figure 3 shows workflow diagram of *DGC*. *Green Compilation* and *Output Generation* modules of ditcc are being modified in order to perform the mentioned tasks. The Energy Cost Statistics and Green Strategies modules require implementation to achieve the functionality of a distributed green compiler.

A C source project is a collection of source code and header files. *DGC* fetches source files from project and pre- processes it by attaching required header files and libraries. Now, either compiler will distribute source code over the network or compile on the same machine depending upon the programmer's choice. Distribution may be performed by distributing source files of project on different slaves for compilation. Each request will be sent to first available machine that will process it. Slaves require running cross compiler and a daemon of *DGC*. In first run of compilation, source code is selected block by block and following *Green Strategies* are applied on selected code:

Loop optimization

Is the process to conserve energy during loop execution. Several techniques of loop optimization are available for example loop tilling, loop fission, loop fusion and loop unrolling. *DGC* performs loop unrolling through *funroll-all-loops* and *fvariable-expansion-in-unroller* switches of GCC compiler. *funroll-all-loops* switch conserves energy by reducing the number of iterations of a loop, while *fvariable-expansion-in-unroller* copies local variables during loop unrolling for dependency elimination.

```
Input:
 1. A C program.
 2. Switches D for distribution
 3. Machine IPs for distribution.
Output:
    1. An energy conservative executable.
    2. Green suggestion.
    3. Energy statistics report of program.
Begin
    while(End of file){
        pre-process source code.}
    If(D){
        Whilse(End of file){
            Distribute on network}
    }
    while(End of file){
        Loop optimization, Dead code and recursion elimination, Un-optimized block
        identification, Energy statistics calculation
        Other Compilation process for intermediate code generation}
    If(D){While(End of file){
                        Integrate}
                }
 Generate output
End
```

Figure 2 Algorithm for proposed Distributed green compiler (DGC).

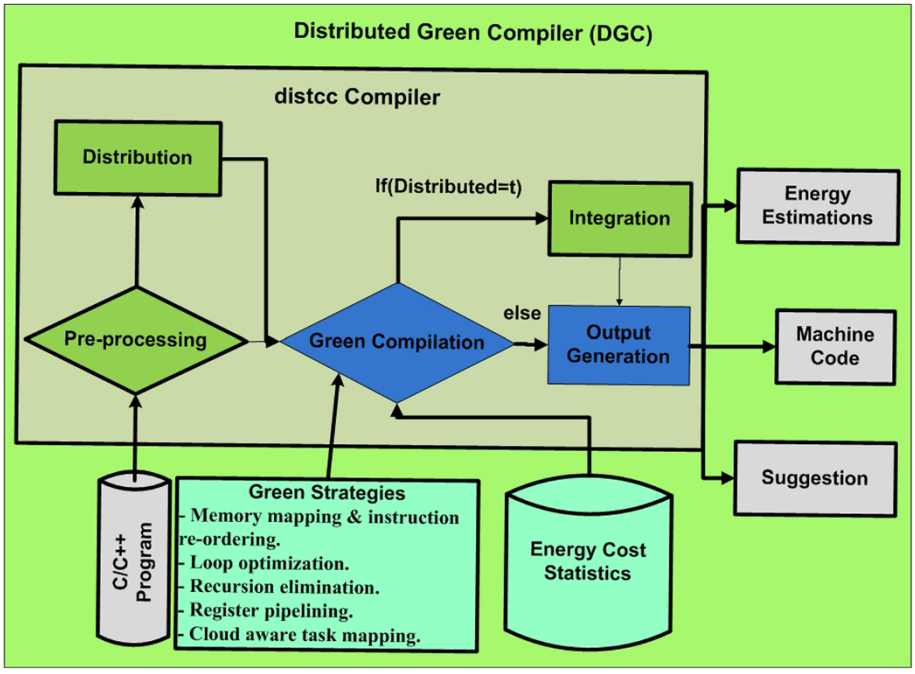

Figure 3 Workflow diagram for proposed Distributed green compiler (DGC).

Use of energy optimized data structure

DGC provides its own energy conservative data structures for software developers.

Dead code elimination

Dead code elimination is the process to remove code, which does not change program result. For example if I have a code that assigns the value to a variable and that variable is assigned the value again, then it is obvious that the first assignment is useless. Dead code elimination will remove the first assignment statement. It shrinks the program and avoids implementing inappropriate operations. It also helps to conserve energy, as executable will not need to process unaffected code and this saves clock cycles at each run of generated executable.

Software pipelining

This technique can be used to optimize loops for overlapping iterations. GCC compiler and some hardware such as Intel IA-64 architecture support this technique. DGC uses GCC compiler support and *Modulo Scheduling* to perform software pipelining that will help DGC to make it hardware independent, as these approaches are hardware dependent.

Recursion elimination

It is not possible to eliminate recursion every time. DGC resolve when it is feasible to convert recursion into iteration. If recursion cannot be converted safely, it offers green suggestion to software developer by highlighting the selected areas of code.

Cloud aware task mapping

DGC uses cluster of physical or virtual machines for distributed compilation process. It uses the hardware and software resources of network to process a compilation problem. Accessing and use of virtualization comes under cloud aware task mapping.

Un-optimized code blocks identification

DGC offers green suggestion to its software developer by highlighting un-optimized block of program. Figure 4 demonstrates that swapping of an array element using memory locations require more low level instructions than a register swap operation [29].

Energy cost statistics

DGC maintains energy cost of each instruction in a database. Different parse tree of selected code is generated using *Energy Cost Statistics* database. Machine code of least energy cost parse tree is generated in final run. Also summary reports will be generated as an output of DGC. In the case of distributed compilation all compiled unit of program are integrated by master machine, and sent to output module for further processing. Output module generates energy optimization summary report, machine code, and green suggestion.

Generated executable is highly optimized machine-readable code. Green suggestions will be highlighted areas of program that is not optimized by compiler and can be energy conserved by implementing green strategies for software developer as discussed in section IV.

This research is only describing concept, high-level workflow and high-level architecture of Distributed Green Compiler. However, implementation details of different green aspects and other components are not part of this research work. We are in the phase of implementing a prototype of this research and all implementation details and experiment results will be the part of next version of this paper. For supporting proposed concept, following is an analysis that made on different patch of code using few green aspects which shows that code can be 40% to 60% energy conserved if implements few green aspects. While, DGC claims to improve this percentage significantly because its implement all discussed green aspects on code at compile time.

Suppose a P program uses V_{cc} supply voltage and I average current to achieve its goals within T seconds, than the total energy E consumed by a program can be calculated with famous known equation [18]: $E = V_{cc} * I * T$.

We can write $T = N_{cc} * \tau$ where N_{cc} is number of clock cycle and $\tau$is the clock period. So energy's equation can be rewritten as: $E = V_{cc} * I * N_{cc} * \tau$.

$V_{cc}$and τ remains same for a specific hardware thus they are considered as constant. By reducing the factor of time a program will be energy conservative, from time

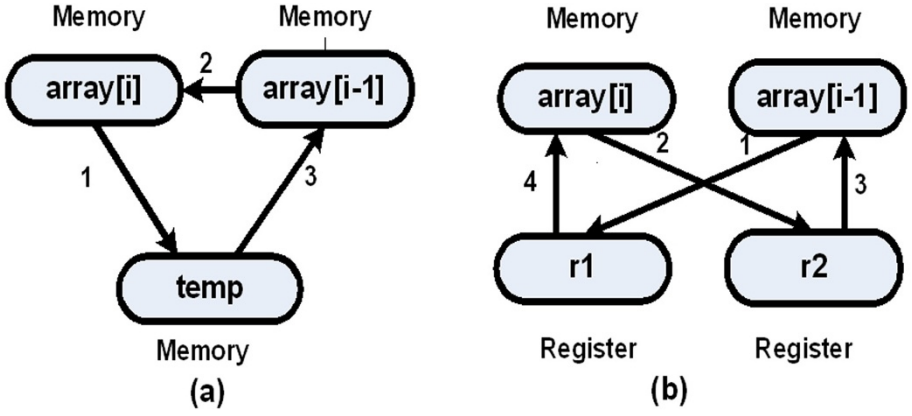

Figure 4 Array element Swapping operation.

equation $T = N_{cc} * \tau$. Sinceτ is assumed as constant so we need to reduce N_{cc} in order to make a program energy efficient.

In general, green compilers take comparatively high compilation time; this problem can be of a little significance in small projects but is a bottleneck for large projects. Energy cost of a block can be calculated by summing up the clock cycle of all instruction within that block [29]. In our DGC prototype model, several green strategies such as l*oop unrolling, source code distribution, use of register operands and recursion elimination* are applied to different source codes as experiment. Comparing the clock cycles of optimized code assembly verses unoptimized code assembly, it was observed that optimized code reduce N_{cc} factor by 30% to 40% as given in Figure 5. Instruction cycles of Intel 80486 family architecture used to calculate the energy cost for this experiment are given in [30].

DGC is a distributed compiler and uses distcc as baseline. Study in [28] shows that distcc is 3.x faster than single machine compilation. If Linux 2.4.19 needs to be compile on a single 1700 MHz P4 machine distcc 0.15 will take only 6 minutes 45 seconds. Across three such machines, using distcc with a 100Mbps switch it takes only 2 minutes 30 seconds that means 89% increase in efficiency.

Table 2 shows the features analysis of DGC with other compilers. The symbol "☼" used in Table 2 shows the fulfilment of particular feature (specified in top row i.e. row 1) by the compiler (specified in leftmost column i.e. column 1) whereas the symbol "×" shows compiler does not cater with the particular feature.

Diet scheduler with green aspect

This section describes the methodology of proposed scheduler with green aspects. The scheduling system used in DIET is at the agent level but it has been observed that it does not take into account the green aspect. For this, some changes are done in the scheduling system of DIET to include the green aspect. To make energy efficient scheduling the greedy based algorithm for efficient VM allocation to processor core has been adapted from [15]. It has been observed that utilization of all cores of processors by VMs reduces the power consumption [16]. For this, greedy based approach from [16] and largest capacity strategy from [15] is adapted in proposed scheduler. Greedy based algorithm ensures

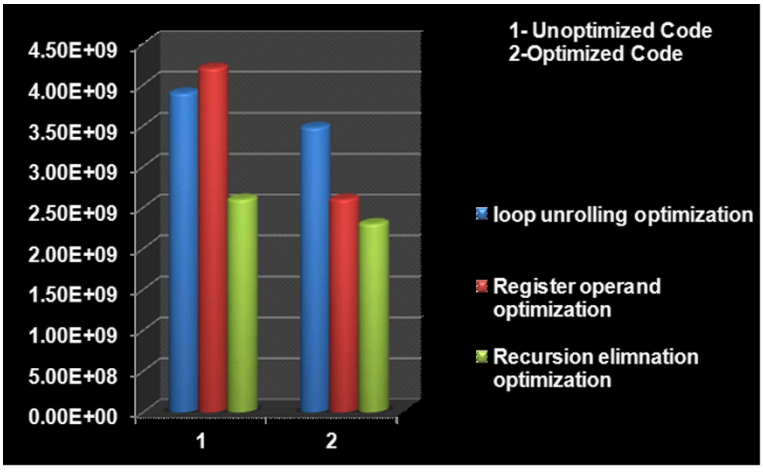

Figure 5 Experiment comparison graph for Intel 80486 family architecture.

Table 2 Features analysis of DGC with existing compilers

Features/Compilers	DGC	Encc	Coffee	Mrcc
Distributed	☼	×	×	☼
Hardware Independency	☼	×	×	☼
Cloud Aware Task aping	☼	×	×	☼
Energy Cost Calculations	☼	☼	☼	×
Loop Optimization	☼	×	☼	×
Dynamic Power Management	×	×	×	×
Instruction Reordering	×	☼	☼	×
Recursion Elimination	☼	×	×	×
Register pipelining	☼	☼	☼	×

the allocation of VM to processing cores in such a way all the cores in nodes are utilized whereas the largest capacity algorithm is for efficient allocation of tasks to VMs.

Client agent submits an application request to master agent (MA), which passes the request to the local agent (LA). The local agent maintains the list of available servers and the VMs running on each server. The reason for including the VMs information is virtualization of one of the efficient techniques for conserving power and minimizing the number of physical machines. For this purpose, some proper scheduling of VMs is required. Based on request type, LA passes the request to server daemons (SeD). The SeD contains the list of problems that can be solved on various VMs, list of available cores in processor, list of available data on machine, and information including CPU capacity and available memory. SeD on receiving the request creates the performance estimation vector that contains the performance estimation values.

Performance estimation values are generated by collector of resource information (CoRI) which gathers the information from the Network Weather System (NWS), FAST (Fast Agent System Timer) and CoRI-Easy. NWS, FAST, and CoRI-Easy are performance prediction tools. FAST predicts the execution time of application and relies on NWS for performance estimation, which provides information like; CPU availability, free memory, number of cores available, and network bandwidth. CoRI-Easy provides the information like; memory capacity, CPU evaluation, network performance, and Disk performance and capacity. Moreover, CoRI manager manages the interaction between different collectors. On receiving the response from various collectors the list of servers are passed to the SeD, which passes the list to the LA. LA sorts the list based on available VMs on a particular server. The machine with large number of VMs is given highest priority. The list (of servers only not VMs) is then given to MA, which passes it to client. Client on receiving the list selects the server for computation. It is to be notified that client can only select the machine from the list but cannot have direct access to any machine.

The selected servers are given to the MA and then to LA. Now LA has the responsibility of distributing the tasks among various servers. Based on number of VMs available at server, a specific server is selected and VM is selected because of capacity. Let us suppose that two servers are selected for computation. Server 1 has large number of VMs and capacity. The VMs are selected in such a way that all processing elements are utilized. Tasks are distributed according to the capacity. The VM with larger capacity is given the largest task (or portion of task if the task is to be distributed) and so on. Now suppose that all the VMs in server 1 are utilized. For next job, server 2 will be selected and rest of the task is distributed

```
Begin
    while (true)
    {
      for( i = 1 to i <= queue1.length( ))
      {
        If  ci >= 1 && queue1.length( )>0)
        {
          If check capacity vm on ci
          {
            Schedule vm on ci AND task on VM
            ci – 1
          }
        }
      }
    }
End
```

Figure 6 Greedy capacity Algorithm.

according to the greedy capacity algorithm. Figure 6 shows greedy capacity algorithm. The list of VMs on machine is maintained in queue with LA and tasks in queue1.

LA maintains the queue of available VMs on machines and another queue let us say queue1 of submitted tasks is also maintained. On selection of server(s) by client, the LA allocates the VM on processing cores according to the capacity in such a way that all processing cores in a machine are utilized. The tasks are allocated according to the capacity of VMs and size of task. If the task cannot run on one machine then the same procedure is followed to allocate the task on number of machines.

The major aim of this algorithm is to utilize all processing cores in a machine, which reduces the power consumption. Figure 7 shows proposed scheduler. It has been observed from the literature that utilization of all processing cores in a machine

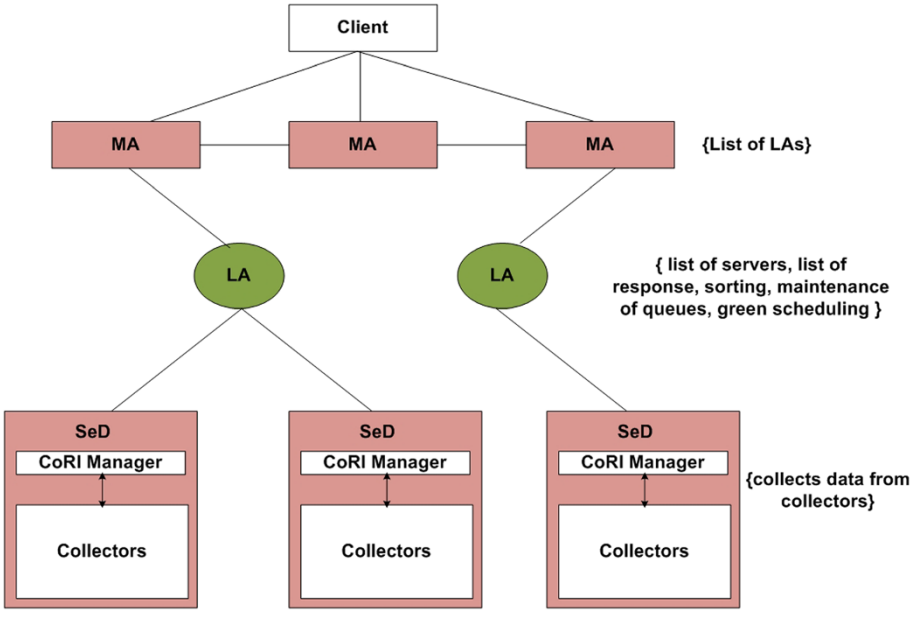

Figure 7 Proposed DIET Scheduler with green aspects.

reduces the power consumption. This approach is used in the proposed scheduler that not only reduces the power consumption but also efficiently utilize the cloud resources.

Conclusion & future work

Green compilation is a software level technique to conserve energy. A green compiler applies several green strategies to reshape source code during intermediate code conversion and generates energy conservative executables. In this paper, several techniques for green compilation are highlighted. This paper also discusses techniques that a software developer can adopt to develop energy conservative programs.

A distributed green compiler is proposed that uses some of the identified techniques at compilation level. However, some source code cannot be reshaped during compilation and is supposed to be handled by software developers. DGC highlights these source code blocks.

Energy conservation and performance are conflicting goals and compilation time of program is increased when green strategies are applied. DGC handles this problem by distributing the program over network of physical or virtual machines. It facilitates software developer by giving the option to compile the program on a single node as well as multiple nodes. Performance analysis shows that DGC conserve clock cycles by 30% to 40% by applying few green strategies. Future work of this paper is detailed performance analysis of DGC with existing compilers, after completing its prototype.

The provision of resources by cloud has added many advantages in terms of saving cost but to efficiently and effectively make use of its resources, a good scheduler is also required. In this research, green aspect is introduced in DIET scheduling so that in addition to efficient utilization of resources, power consumption and carbon dioxide emission could be reduced.

Competing interests

The authors declare that they have no competing interests.

Authors' contributions

FF carried out research on energy conservation techniques on software and hardware level. She proposed the idea of Distributed Green Compiler for software level Green Computing. She participated in alignment and preparation of menu script. BJ carried out research on scheduling techniques for energy conservation. She gave the idea of Green Scheduler. RUR participated in both researches by improving proposed ideas. He participated in proof reading and improving the menu script. OM participated in proof reading of menu script. KZ participated in proof reading of menu script. All authors read and approved the final manuscript.

Acknowledgment

The author would like to thank Mr. Rauf ul Hassan for his help to review menu script and his valuable discussion on Distributed Green Compiler.

Author details

[1]School of Electrical Engineering & Computer Science, National University of Science & Technology, Islamabad, Pakistan. [2]Techaccess NUST Research & Development Center, Islamabad, Pakistan.

Received: 30 December 2011 Accepted: 11 April 2012
Published: 22 May 2012

References

1. Tiwari V, Malik S, Wolfe A (1994) Compilation Techniques for Low Energy: An Overview. J Low Power Electrons – JOLPE IEEE Symp, 38–39
2. Bellas N, Hajj IN, Polychronopoulos CD, Stamoulis G (2000) Architectural and Compiler Techniques for Energy Reduction in High-Performance Microprocessors. IEEE Trans Large Scale Integration (Vlsi) Syst 8(3):317–326
3. Freescale Semiconductor, Inc (2011) EXtreme Energy Conservation, Advanced Power-Saving Software for Wireless Devices. White paper Freescale Semiconductor, Inc. [http://www.freescale.com/files/32bit/doc/white_paper/XTMENRGYCNSVWP.pdf]
4. Heath T, Pinheiro E, Hom J, Kremer U, Bianchini R (2002) Application Transformations for Energy and Performance-Aware Device Management. International conference of Parallel Architectures and Compilation Techniques, pp 121–130

5. Kremer U, Department of Computer Science Rutgers University (2002) Low Power/Energy Compiler Optimizations. International conference of Book power aware computing. CRC Press, 2004, ch. 35
6. Jaeger PT, Lin J, Grimes JM, Simmons SN (2009) Where is cloud? Geography, Economics, Env Jurisdiction Cloud Comput 14(5)
7. Plug in Scheduler. retrieved on 7th January, 2011, [http://graal.ens-lyon.fr/DIET/scheduling.html]
8. A compiler framework for the reduction of worst-case execution times., Retrieved on 3rd July, 2011, [http://ls12-www.cs.tu-dortmund.de/research/activities/encc/]
9. Raghavan P, Lambrechts A, Absar J, Jayapala M, Catthoor F, Verkest D (2008) COFFEE: COmpiler Framework for Energy-Aware Exploration. HiPEAC'08 Proc 3rd Int Conference High Perform Embedded Architectures Compilers 4917:193–208
10. Cloud Computing., Retrieved at 03. June 2011 [http://fclose.com/b/cloud-computing/article/mrcc-a-distributed-c-compiler-system-on-mapreduce/]
11. Schopf JM (2003) Ten actions when Grid scheduling: The user as Grid scheduler. Grid Resource Management: State of the Art and Future Trends, ch. 2. In: Nabrzyski J, Schopf JM, Weglarz J (eds) Kluwer Academic, Boston, MA
12. Park SM, Kim JH (2003) Chameleon: A Resource Scheduler in a Data Grid Environment. In: the proceedings of the 3rd IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGrid 2003)., Tokyo, Japan, pp 258–265
13. Ayyub S, Abramson D (2007) GridRod - A Service Oriented Dynamic Runtime Scheduler for Grid Workflows. In: Proceedings of the 21st annual international conference on Supercomputing (ICS '07). ACM, New York, NY, USA, pp 43–52. doi:10.1145/1274971.1274980
14. Krueger P, Babbar D (1993) Stealth: A liberal approach to distributed scheduling for networks of workstations. Technical Report, OSUCISRCI/93-TR6. Ohio State University
15. Duy TVT, Sato Y, Inoguchi Y (2011) A prediction-based green scheduler for datacenters in clouds. IEICE Trans Inf Syst E94-D(9):1731–1741
16. Ishfaq Ahmad (2012) Handbook on Energy-Aware and Green Computing., University of Texas at Arlington, USA; Sanjay Ranka, University of Florida, Gainesville, USA
17. Tang Q, Gupta SKS, Varsamopoulos G (2008) Energy-efficient thermal-aware task scheduling for homogeneous high-performance computing data centers: a cyber-physical approach. IEEE Trans on Parallel and Distributed Systems 19(11):1458–1472
18. Hsu C, Feng W (2005) A power-aware run-time system for high-performance computing. In: Proceedings of the 2005 ACM/IEEE conference on Supercomputing (SC '05). IEEE Computer Society, Washington, DC, USA, p 1 doi:10.1109/SC.2005.3
19. Al-Fares M, Radhakrishnan S, Raghavan B, Huang N, Vahdat A (2010) Hedera: Dynamic Flow Scheduling for Data Center Networks. Hedera: dynamic flow scheduling for data center networks. In: Proceedings of the 7th USENIX conference on Networked systems design and implementation (NSDI'10). USENIX Association, Berkeley, CA, USA, p 19
20. Stage A, Setzer T (2009) Network-aware migration control and scheduling of differentiated virtual machine workloads. In: Proceedings of the 2009 ICSE Workshop on Software Engineering Challenges of Cloud Computing (CLOUD '09). IEEE Computer Society, Washington, DC, USA, pp 9–14. doi:10.1109/CLOUD.2009.5071527
21. Kliazovich D, Bouvry P, Khan SU (2010) DENS: Data Center Energy-Efficient Network-Aware Scheduling. IEEE-ACM International Conference on Green Computing and Communications and International Conference on Cyber, Physical and Social Computing. IEEE/ACM Int'l Conference on Green Computing and Communications & Int'l Conference on Cyber, Physical and Social Computing, pp 69–75
22. Naik K (2010) A Survey of Software Based Energy Saving Methodologies for Handheld Wireless Communication Devices. Tech. Report No. 2010-13. Dept. of ECE, University of Waterloo
23. Su C-L, Tsui C-Y, Despain AM (2002) Low Power Architecture Design and Compilation Techniques for High-Performance Processors. Compcon Spring '94, Digest of Papers, pp 489–498
24. Delaluz V, Kandemir M, Vijaykrishnan N, Irwin MJ (2000) Energy-Oriented Compiler Optimizations for Partitioned Memory Architectures. CASES '00 Proceedings of the 2000 international conference on Compilers, architecture, and synthesis for embedded systems
25. National Compiler Infrastructure (NCI) project., Retrieved overview at 03 June 2011, from [http://www-suif.stanford.edu/suif/nci/index.html. Co-funded by NSF/DARPA]
26. Heath T, Pinheiro E, Hom J, Kremer U, Bianchini R (2002) Application transformations for energy and performance-aware device management. Parallel Architectures and Compilation Techniques
27. Mehtal H, Owens RM, Irwin MJ, Chen R, Ghosh D (1997) Techniques for Low Energy Software. In: Proceedings of the 1997 international symposium on Low power electronics and design (ISLPED '97). ACM, New York, NY, USA, pp 72–75. doi:10.1145/263272.263286
28. Google Code., Retrieved on 03, June 2011 from [http://distcc.googlecode.com/svn/trunk/doc/web/index.html.]
29. Naik K, Wei DSL (2001) Software Implementation Strategies for Power-Conscious Systems. J Mobile Networks App 6(3)
30. 80×86 instruction set., Retrieved on 03, June 2011 from [http://www.penguin.cz/~literakl/intel/intel.html]

doi:10.1186/2192-113X-1-4
Cite this article as: Fakhar *et al.*: **Software level green computing for large scale systems.** *Journal of Cloud Computing: Advances, Systems and Applications* 2012 **1**:4.