GA Assisted SDF Scheduling for Energy-Aware Mapping of Heterogeneous Processors

Nicholas Kelly, Christopher Erb

***Abstract*—We present a method of optimizing both energy and latency of a Synchronous Dataflow model (SDF) using genetic algorithm (GA) assisted list scheduling. Our solution is fully automatic requiring only the SDF definition, processor specifications, and communication bus specifications. Using these parameters, latency and energy are accurately modeled taking into account dynamic energy, static energy, and DVFS overhead. Actor firings are then optimized for minimal latency and energy using SPEA2, a multi-objective GA. The resulting schedule defines the overall timing schedule, DVFS schedule, and processor mapping. In initial tests, energy-optimized schedules can consume up to 38.5% less energy, with a minimal effect on latency. Our final system is fully configurable and versatile, working on SDFs (for DSP applications) and task graphs (for general applications).**

I. Introduction

Due to the power-wall seen in homogeneous processors, modern embedded systems have been transitioning to heterogeneous, multiprocessor systems. Additionally, with the popularity of mobile devices, energy has emerged as a constraint, especially for todays battery technologies. In order to minimize energy, a variety of power-saving techniques are applied. Big-little architectures contain processors of varying performance in order to save energy on less intensive tasks, while still maintaining responsiveness. Dynamic Voltage and Frequency Scaling (DVFS) is employed in some processors allowing the operating voltage (and/or frequency) to be scaled in relationship to the performance needed. Both of these techniques allow for significant energy savings in certain situations; however, the method of scheduling and mapping tasks in order to optimize for energy is challenging in most situations. The search space of this problem is very vast, due to the flexibility offered by the different processors, concurrent tasks, and energy-saving techniques (DVFS). Such a problem is considered NP-Hard and requires methods for generating quality (not necessarily optimal) solutions while optimizing energy and execution time.

For this reason, various models of computation (MoC) have been created which allow for concurrency, determinism, and ease of scheduling. We have chosen to consider Synchronous Dataflow (SDF) graphs, attributable to their properties which allow for static (offline) scheduling. A SDF is a graph with various nodes (actors) which are connected by arcs. Each actor requires and produces a static amount of data (tokens) which allows for a periodic schedule of firings. SDFs are primarily used for modeling DSP systems, for applications in image processing, audio decoding, etc. However, task graphs (Homogeneous SDF), a subset of SDFs, can be used for modeling general programs where each task represents a basic block (BB). For this reason, SDFs are an important, general MoC which can be used to statically allocate, map, and schedule tasks to various processors.

In this paper, we propose an energy-aware process of mapping and scheduling using SDF graphs, while maintaining timing and cost constraints. We choose SDF graphs because they have a diverse usage, and are able to be scheduled offline, allowing for more energy savings. In order to find optimal solutions, we propose a genetic algorithm (GA) which maps and schedules each actor instance on a set of processor elements (PEs). While genetic algorithms are usually slower in execution time, they have been shown to avoid local-minima and yield more optimized solutions. Additionally, they allow for multiple optimizations through a tuned, multi-parameter fitness function or multi-objective GAs. Our GA contains 3 steps (mutations): mapping (actor-processor assignment), scheduling (general actor schedule, list schedule), and runtime scheduling (specific time actor runs).

There are several benefits to our solution: first, the input to the program is only a SDF graph and a list of processors with timing, power, and cost characteristics. Furthermore, since GAs are iterative, a simulator could potentially be run to measure timing and power statistics instead of being specified in the input. Second, SDFs are applicable in both DSP and general processing environments, allowing for diverse usage. Third, it uses a GA to produce quality, optimized mappings and schedules based upon energy, timing, and cost within a specified amount of acceptance.

II. Related Work

A. SDF Scheduling

SDFs are first proposed in [1], where scheduling methods are described. For our purposes, we adapt methods for generating the repetition vector for the SDF which aids in creating a precedence graph for scheduling. Unlike other techniques, we are able to only require the SDF as input, instead of the repetition vector or other parameters. While we could generate a precedence graph (HSDF) like other solutions [2], we instead decided to use the repetition vector, along with the SDF graph to generate multiprocessor schedules, similar to [3]. This decision is primarily made for its simplicity since storing a precedence graph with multiple arcs could be complex for increasingly concurrent SDFs with many actors.

Pipeline scheduling with SDFs is described in [2] where execution is pipelined with the next iteration of the SDF running at the same as the current iteration. While this

increases the throughput, it means there is effectively little idle time on processors allowing for less opportunities for DVFS, which is why we ignore it for our solution. However, it should be noted that since the throughput is increased through pipelining, the effective energy per iteration is decreased, thus it could yield interesting future work. Multi-Frequency SDF graphs are described in [4] where different sections of the SDF are assigned a certain frequency, which is especially useful for DSP (sampling) applications. In their solution, actors are assigned to bins based upon operating frequency. In our case, we want to keep SDF general since the same functionality could be replicated in the actors instead of scheduling. However, since bins can have significant idle time, multi-frequency could make a useful addition for DVFS utilization.

B. Genetic Algorithms for Scheduling

Once the dependencies of the actors are known, the problem becomes exploring the expansive space of processor mappings and schedules. To explore this space, we will be using genetic algorithms. The genetic algorithm approach towards mapping and scheduling is most similar to [5]. However, instead of having a two-step iterative process, we specify mutation rates for each level in our GA, where mapping has the lowest mutation rate, scheduling has the second lowest, and runtime scheduling has the highest. Having multiple mutation rates allows parameters to change according to their granularity, allowing for the different steps to be execute in a single GA. [6] uses a strategy that singly replaces individuals with child schedules to lessen the effect of population size and remembers the best schedule, regardless of it being in the population pool. Similarly, in order to keep a diverse population (no local-minima), we allow the fittest individual to be replaced, though the individual still be archived. The idea of a cluster presented in [7] promotes solution diversity by basing the evaluation of fitness between significantly different populations off of a looser metric. We also implement this in order to help avoid local-minima using a island-model of GAs.

C. Optimizing for Energy

The first layer of the energy problem is the process mapping and scheduling which will have effects on energy consumption due to inherit processor properties. Once a scheme has been discovered, a popular tactic is to use voltage scaling along with frequency reduction to utilize slack time and conserve even more power as seen in [5], [8], [9]. As with all GAs, having a well tuned fitness function is crucial, for this reason we used previous solutions as a starting point. Since [5] uses a GA for energy optimization, their fitness function is applicable in our situation. In their solution they have a fitness function for scheduling based upon energy dissipated and the associated time penalty for the schedule. Additionally, they specify a function for mapping which is similar to scheduling but takes PE area into consideration. Our technique follows their conventions, but uses a general cost parameter for each PE instead of area and uses a combined fitness function.

III. Methods

A. Energy Modeling

Our energy modeling is based off of capacitive switching and static leakage of CMOS circuits [10]. While our energy model can be used with absolute (processor accurate) values, it can also be used with relative values, since only comparisons are made in the optimization algorithm (unless an accurate energy result is required). As voltage is scaled, the energy used in different actor firings will change. In order to model the dynamic energy contribution, we use:

$$E_{dyn} \propto a_{0\to1} C_{dyn} V_{dd}^2 N \tag{1}$$

Actor activity factor ($a_{0\to1}$) and execution duration (N, in cycles) are actor specific to the PE. The activity factor describes what percentage of the the PE/components are being utilized during the actor firing. Additionally, the execution duration is the time it takes for an actor firing to run on a certain PE. Dynamic capacitance (C_{dyn}) and V_{dd} are PE specific, where V_{dd} is scaled during runtime to one of a number of discrete voltages specified for the PE. Discrete voltages are used for scaling since continuous DVFS is currently not available many modern processors. However, as explained in the next section, the duration can be scaled continuously by switching the voltage at different times. For modeling static energy, a leakage current needs to be specified:

$$E_{stat} \propto I_{leak} V_{dd} \tag{2}$$

Depending on the processor, the voltage could be the current voltage specified by DVFS, or the core (max) voltage. In combination with processor energy modeling, bus energy is also modeled per token transmission. In this case, a fixed energy value is associated with each cycle tokens are transmitted (dependent on bus frequency). Additionally, our model takes into account DVFS overhead factors such as the PLL lock time and upscaling energy cost [11]. When a DVFS switch occurs where the ending voltage ($V_{dd,e}$) is higher than the starting voltage ($V_{dd,s}$), the following is added to the total energy:

$$E_{dvfs} \propto \begin{cases} C_{dyn}(V_{dd,e}^2 - V_{dd,s}^2) & \text{if } V_{dd,s} < V_{dd,e} \\ 0 & \text{otherwise} \end{cases} \tag{3}$$

Where the DVFS transition (in cycles) is processor specific. Similarly, during any DVFS switch, the duration of the firing currently running is extended by the PLL lock time (where the process is currently disabled) which is also PE specific.

B. Latency Modeling

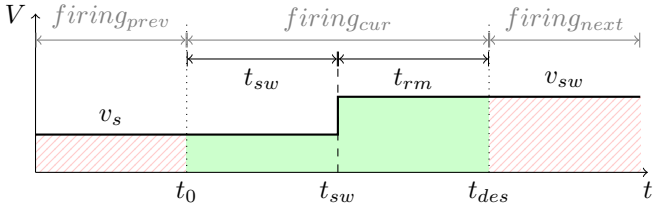

Fig. 1: Actor firing DVFS switching

Our latency modeling (see Figure 1) uses a method similar to [8] where a firing has start voltage (v_s) (set by the previous firing) and an end voltage (v_{sw}, becomes v_s for next firing). The switching point (t_{sw}) not only determines energy savings, but also the overall runtime of the firing. Thus, a firing can be scaled to fit a variety of durations with only a few discrete voltages, allowing for idle slack to be easily recovered. When actors are specified, only the execution time (t_{d0}) at the maximum voltage (V_{max}) for the processor needs to be included, other additional runtimes are calculated based upon the switch time and DVFS voltages (see Equation 4).

$$t_d = \frac{t_{d0} V_{dd} (V_{max} - V_{th})^2}{V_{max} (V_{dd} - V_{th})^2} \tag{4}$$

Since the genotype only modifies the target duration (t_{des}) and switching voltage, the actual switching time of the voltage needs to be calculated. Using Equation 5 and Equation 4, the switching point can be determined (see Equation 9). Note, that during runtime, the switching voltage and target duration values chosen by the GA may lead to an invalid result. To prevent this, multiple voltages are iterated through to try achieve the desired target duration at a different voltage if necessary.

$$t_{des} = t_{sw} + t_{rm}, 1 = \frac{t_{sw}}{t_{d,sw}} + \frac{t_{rm}}{t_{d,rm}} \tag{5}$$

$$V_{ms} = (V_{max} - V_{th})^2 \tag{6}$$

$$V_{cs} = V_{max} (v_s - V_{th})^2 \tag{7}$$

$$V_{sw} = V_{max} (v_{sw} - V_{th})^2 \tag{8}$$

$$t_{sw} = \frac{v_s v_{sw} t_{d0} V_{ms} - t_{des} V_{sw} v_s}{V_{cs} v_{sw} - V_{sw} v_s} \tag{9}$$

C. List Scheduler

A list scheduler is used to schedule actor firings based upon input from the GA. In our implementation the GA makes the majority of high-level scheduling decisions. Thus, the list scheduler is primarily used as a method of resolving possibly invalid schedules. Since the GA generates random, absolute start times, using a list scheduler with a priority list sorted by relative start time guarantees control as well as validity. In addition to time, a global energy for the schedule is kept and updated each time a firing is scheduled (using the previous described energy models). During each timestep, the list is iterated in sorted order and any actors which can currently run on their assigned PE (by GA) are scheduled to run starting at the current timestep. When a firing is assigned to a processor and time, the end time and voltages for the firing are calculated from the target duration (modified by GA). These end times are added to another sorted list which is used to progress time. When a firing ends tokens are sent along the assigned bus (by GA) for the receiving actor, effectively modeling communication and delaying actor start time. Modeling communication introduces additional delays, especially when few buses are available.

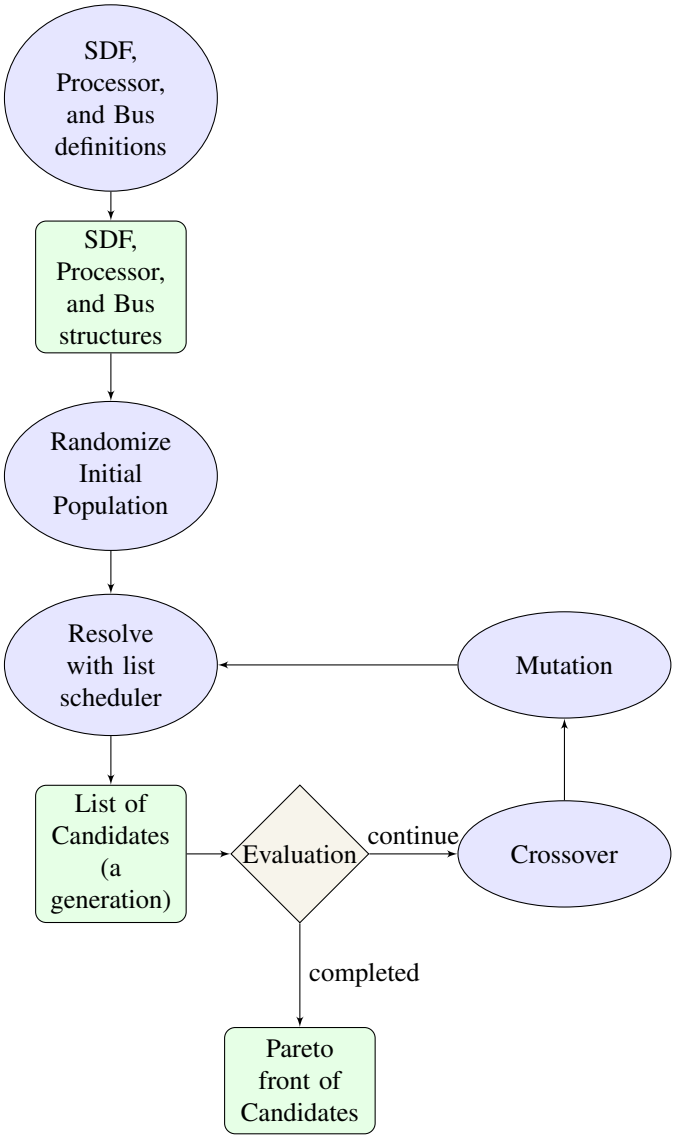

Fig. 2: Program flow

D. Implementation

Since our goal is to optimize both latency and energy, a multi-objective GA is used (see Figure 2). In particular, we use SPEA2 since it has been shown to generate slightly more diverse results and avoid local minima [13] (allowing for more flexible design decisions). In order to initialize the genotype for the GA, we first calculate the repetition vector for the SDF, using a method similar to [1]. The repetition vector effectively defines size of the genotype, where each actor firing could be thought of as a *chromosome* and each attribute of the firing would be a *gene*. A chromosome is composed of $4 + n$ genes where n is the number of actors. These include: processor, start time, target duration, voltage, and bus assignments for sending tokens to each actor. The processor and bus assignments are only modified by the GA, and set during population initialization, crossover, and mutation. Start time is used as the relative priority in the list scheduler, where the actual start time is updated after the resolution phase.

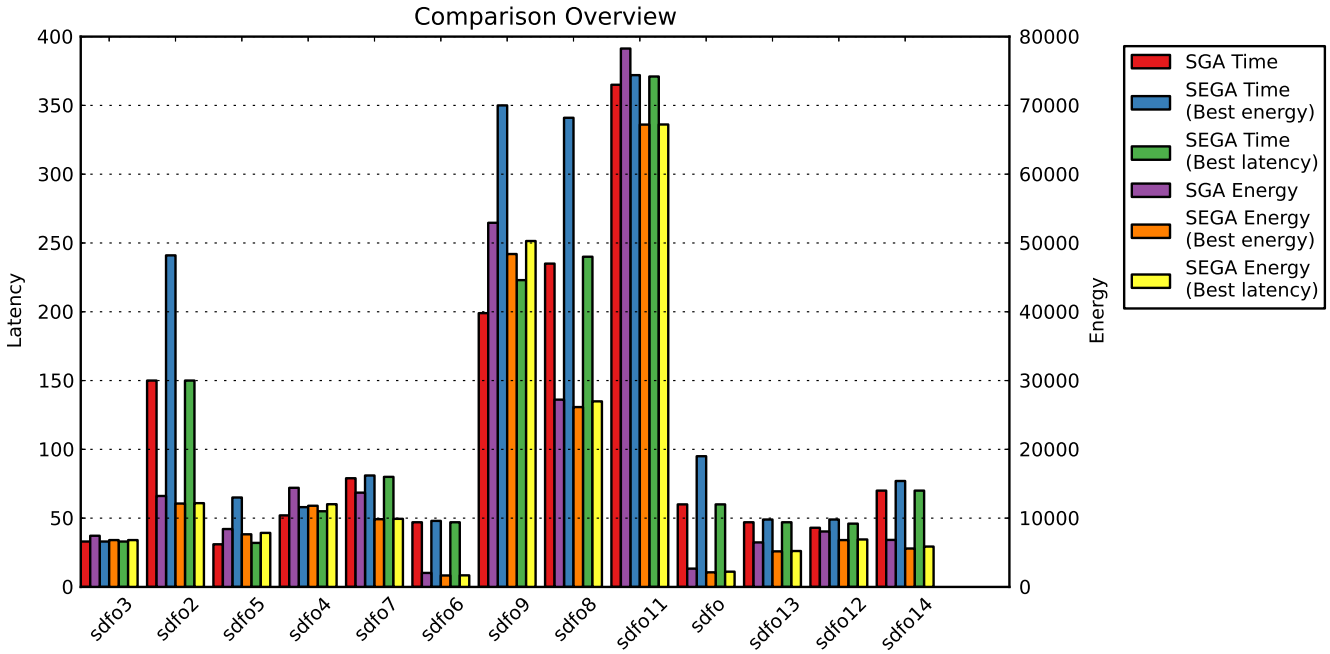

Fig. 3: Simulation results

Similarly, the target duration and voltage are suggestions to the list scheduler in order to calculate firing duration; however, in some cases the target duration may not be possible with the specified voltage and the genes will be updated accordingly after the resolution phase. It is important to note that since a multi-objective GA is used, multiple (pareto) solutions are returned. The appropriate solution to the user's application should be selected or determined with a weighted fitness function.

Due to the complex nature of the optimization problem, we define custom genetic operators which closely resemble bit-string methods but have inherent bounds. In the case of crossover we use a uniform method since it can produce more diverse results and better explore the search space [14]. In our implementation, two offspring are created from two parents, each with random and opposite chromosome selections from either parent (which are resolved with the list scheduler). Additionally, our mutation function defines multiple levels of mutation (through weighted probabilities) in order to have tight control over how often various genes change. For example since firing schedules are dependent on processor mapping, processor assignment mutations should occur less often (effectively having a lower rate). Our mutation function modifies multiple genes corresponding to their contextual (based upon other genes) bounds, and finally resolving validity through the list scheduler.

Additionally, in order to simplify the modification and usage of genetic information, we classify each individual as a *Candidate* and each actor firing as a *Firing*. The genetic operators described operate directly on these objects and then translate the genetic information into the defined genotype structure. Additionally, each of these classes contain methods for data manipulation (e.g. energy calculation) through using the SDF configuration data as a reference.

TABLE I: Test results and baseline comparison

Test	Runtime	Gen.	Latency improv.	Energy improv.
sdfo	137.4s	27226	0.0%	20.1%
sdfo2	821.1s	21335	0.0%	8.6%
sdfo3	1063.9s	20002	0.0%	9.2%
sdfo4	3669.6s	20002	-5.5%	19.8%
sdfo5	1374.2s	24344	-3.1%	7.2%
sdfo6	155.7s	22860	0.0%	20.9%
sdfo7	1304.8s	20254	-1.2%	38.5%
sdfo8	1815.2s	20004	-2.1%	0.9%
sdfo9	5571.0s	20002	-10.8%	5.3%
sdfo11	6214.5s	20056	-1.6%	16.4%
sdfo12	914.1s	21803	-6.5%	16.8%
sdfo13	593.5s	20108	0.0%	24.0%
sdfo14	976.9s	23365	0.0%	16.5%

IV. Results

Our final implementation was done in C++ using SPEA2 from the ParadisEO library for multi-objective searching using genetic algorithms [15]. A configuration file (YAML) is the main input to the program which contains 3 matrices representing the tokens consumed by each actor on a edge, tokens produced by each actor on a edge, and initial tokens on a edge. The file also contains information about each processor (e.g. voltage values) and bus available for use (e.g. energy and frequency). Additionally, many aspects of the algorithm are configurable including GA parameters, and optimization variables. For our tests, we refer to only latency optimization as *SGA* (SDF GA), and energy and latency optimization as *SEGA* (SDF Energy GA).

To generate test graphs, we use SDF3 [12] which is a command line and C tool for randomly generating SDF graphs.

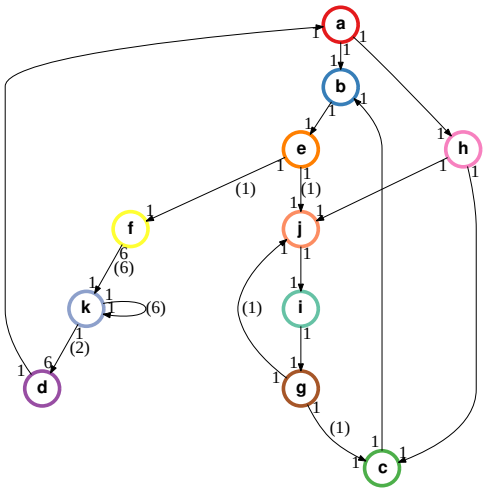

Fig. 4: *sdfo3* SDF graph (initial tokens in parentheses)

It allows the user to define several parameters that control the size and shape of the graph, and then randomly generates a graph based off of this input. These non-trivial graphs are converted to a configuration file for our program which can produce an optimal schedule for completion time or optimal energy/latency schedule. However, there are some attributes which are not generated by the tool. In these cases, we have used uniform and partially random values for processors and buses. Additionally, the parameters for the energy model have been simplified by setting C_{dyn} to 1 and setting E_{stat} to a percentage of the max E_{dyn}. Furthermore, generating an optimal energy schedule involves more decision variables (larger genotype) and as a result takes more generations to resolve for the GA.

We have run the tool on 13 randomly generated SDFs. All tests were performed on a 2.53GHz, dual core system with 4 GB RAM. Additionally, the program was set to run on a population of 20 individuals for a minimum of 10,000 generations with additions of 10,000 until no improvement was detected.

In Figure 3 we have run time and energy for 3 points. The first is scheduling optimization without energy consideration (baseline), the second is with energy consideration and optimizing energy, the third is with energy consideration and optimizing run time. For nearly every test, our energy-optimization shows improvement over the baseline. However, for the best-energy results in the pareto solutions for each test, the latency can sometimes be considerably higher. In order to show the effectiveness of our algorithm with minimal change to latency, we select the best-latency case for following diagrams.

In Table I we show the percentage differences of the energy optimizations when compared to the latency optimizations (baseline). For these initial tests, there is minimal change to the baseline latency. Additionally, most tests show a significant improvement in energy consumption. However, it should be noted that the effectiveness of the algorithm is completely dependent on the SDF. And in some of our testcases, the baseline schedule showed little or no areas for energy-optimization.

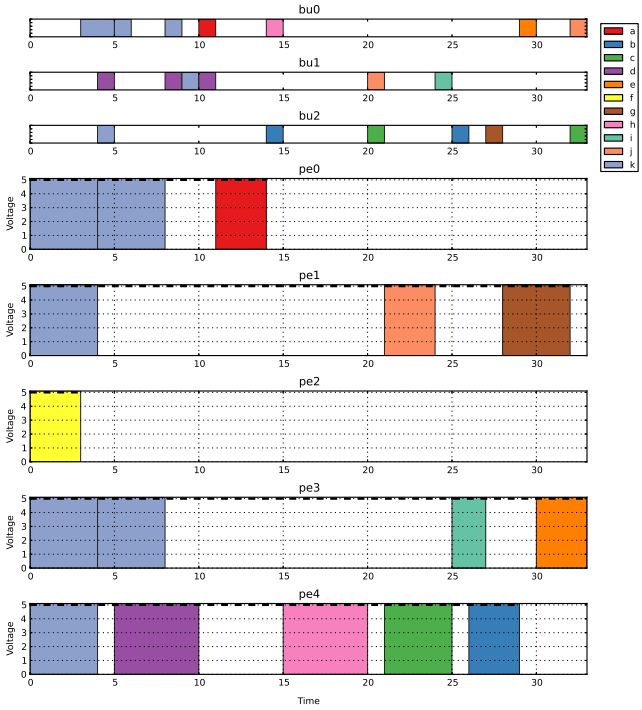

Fig. 5: *sdfo3*, with no energy consideration

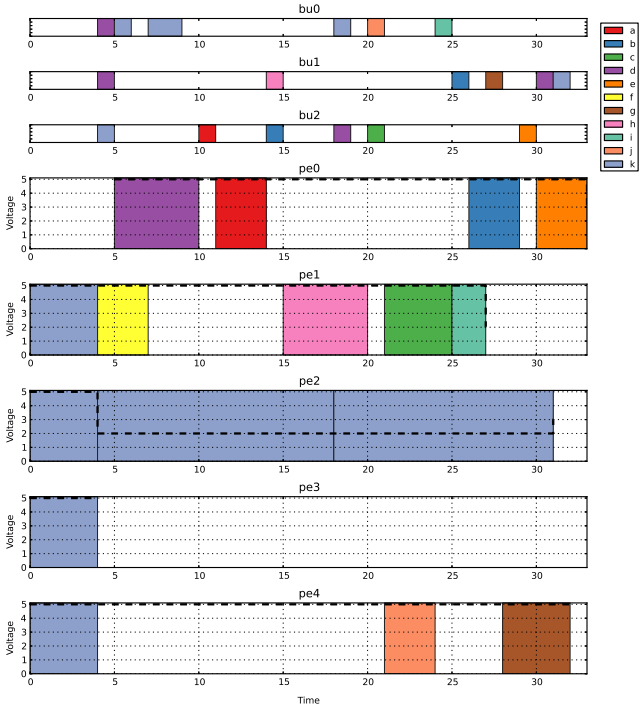

Fig. 6: *sdfo3*, with energy optimization using DVFS

Additionally, we present *sdfo3* and *sdfo4* as example results from our algorithm.

Figure 5 is a runtime chart output of *sdfo3* from our program. This SDF was input with the assumption that there were 3 buses and 5 processors. The 3 narrow bars represent bus communication, with the bar-color representing the receiving actor. Each of the sub-charts represent the timeline for one processor, with each bar-color representing the running actor. From this graph it can be seen that no unnecessary delays are present. No processor is idle when there is a process ready to run, and the process that has the most dependencies is run first. This means it is an optimal (or close to optimal) solution.

Once energy consideration is added, the outcome should be that any process not on the critical path will be elongated by running at a lower voltage. Note, that this is because for our timeline graphs we choose the pareto solution which yields the lowest latency. In Figure 6, this can be seen to be true, where the dashed black line represents the current voltage of the processor. Since the later firings of actor k are not on the critical path (refreshing initial tokens for next cycle), their runtime can be extended by lowering the voltage. Additionally, *sdfo4* has been provided as an example exhibiting similar results in Figure 7, Figure 8, and Figure 9.

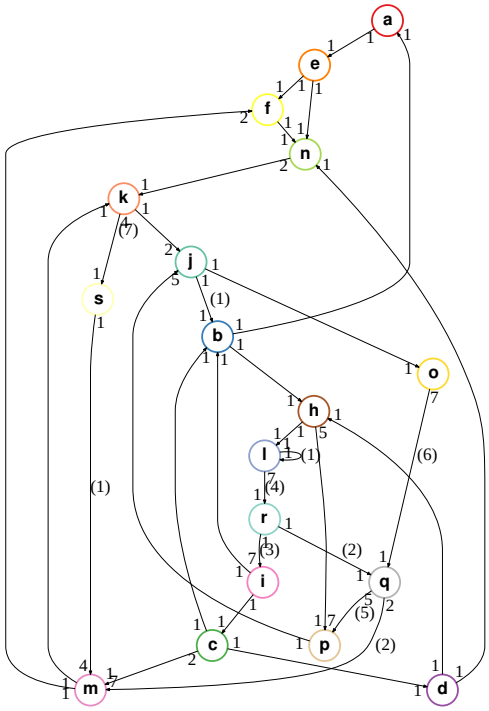

Fig. 7: *sdfo4* SDF graph

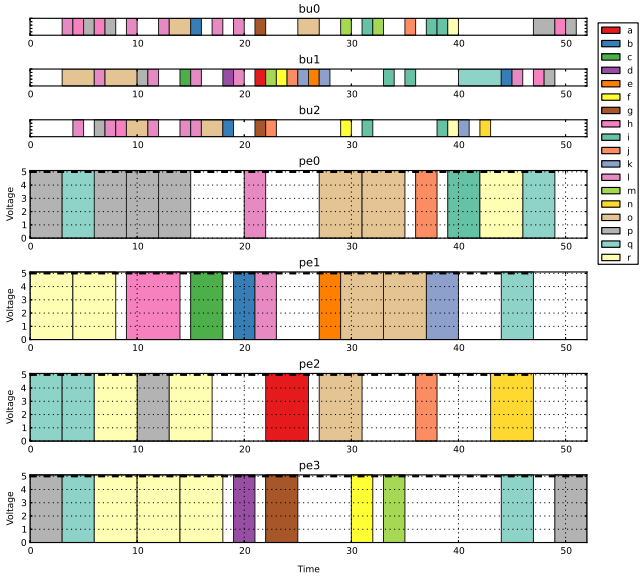

Fig. 8: *sdfo4*, with no energy consideration, running with the assumption of 3 buses and 4 processors, hence 3 bars and 4 sub-charts.

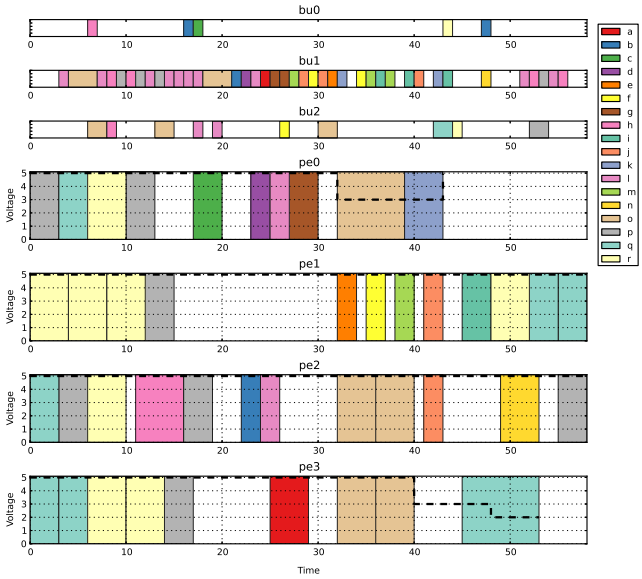

Fig. 9: *sdfo4*, with energy optimization using DVFS

V. Innovation

We have created a new method for determining a sub-optimal mapping and schedule for a SDF based upon various PE and bus parameters. We have used a unique combination of various methods, including list scheduling, genetic algorithms, and accurate energy modeling with DVFS overhead. Additionally, our final tool is configurable, allowing for diverse architectures and many modifications to the optimization flow. Finally, since our system supports SDFs, it is applicable in both DSP and general processing environments (task graphs).

VI. Future Work

Currently, our initial tests have contained arbitrary representations for processors and buses. While this provides a proof of concept, these representations will eventually have to be expanded, made more explicit, and tunable to different processors and buses. Similarly, our final program and process

needs to be further tested on a more diverse grouping of SDFs, in order to understand its optimization effectiveness. Furthermore, creating an ILP (integer linear programming) in order to find optimal solutions so that the accuracy of the program can be determined.

Presently, the bus communication module is not capable of differentiating between processors, and as a result a bus is used for every token produced. To improve the tool this processor awareness will have to be added for this feature. We think that this will be solvable with a two-pass list scheduler. Taking unnecessary tokens off the bus will deflate run times and improve accuracy, as well as allowing for buses to only be connected to specific processors.

Additionally, in order to increase the convergence rate, DVFS capabilities could be added to the list scheduler. The GA normally does this randomly given enough time, but if we add this capability to the scheduler, we can more intelligently explore the problem space and the optimal individuals in the local area should be reached more quickly. Similarly, firing preemption could be added which would allow for firings to better fit in to idle periods (with the comination of DVFS). However, we are unsure if improving the list scheduler (since it is used as a resolution function) will cause the program to be attracted to local minima. Pipelining could be another improvement which could be added to the system, allowing for decreased cycle latency, but possibly less opportunity for energy-optimization. Finally, in terms of the program, support for multiprocessor and distributed environments would be useful in order to decrease the runtime for finding optimal results.

VII. Conclusion

Heterogeneous processor systems are increasingly a critical addition to today's mobile, energy-constrained devices. However, few gains can be achieved without optimal mapping of processes to appropriate processors. In this paper we have presented a method for optimizing a SDF schedule and mapping in terms of latency and energy. Our method uses accurate latency and energy models, allowing for DVFS on multiple processors. Additionally, communication between actor firings is considered and allocated to various buses. In our initial tests, we have recorded promising results of up to 38.5% improvement in energy consumption with minimal effect on the overall latency of the schedule.

References

[1] E. Lee and D. Messerschmitt, "Static Scheduling of Synchronous Data Flow Programs for Digital Signal Processing," *IEEE Trans. Computers*, vol. 36, no. 1, pp. 24-35, Jan. 1987.

[2] D. Li and J. Wu, "Energy-Aware Scheduling for Acyclic Synchronous Data Flows on Multiprocessors," accepted to appear in *Journal of Interconnection Networks*.

[3] Khasawneh, Samer. "Static Scheduling for Synchronous Data Flow Graphs." Electronic Thesis or Dissertation. University of Akron, 2007. *OhioLINK Electronic Theses and Dissertations Center*. 27 Mar 2014.

[4] Knerr, B.; Holzer, M.; Rupp, M., "Task Scheduling for Power Optimisation of Multi Frequency Synchronous Data Flow Graphs," *Integrated Circuits and Systems Design, 18th Symposium on*, pp. 50,55, 4-7 Sept. 2005.

[5] Schmitz, M.T.; Al-Hashimi, B.M.; Eles, P., "Energy-efficient mapping and scheduling for DVS enabled distributed embedded systems," *Design, Automation and Test in Europe Conference and Exhibition, 2002. Proceedings*, pp. 514,521, 2002.

[6] Grajcar, M., "Genetic list scheduling algorithm for scheduling and allocation on a loosely coupled heterogeneous multiprocessor system," *Design Automation Conference, 1999. Proceedings. 36th*, pp. 280,285, 1999.

[7] Dick, R.P.; Jha, N.K., "MOGAC: a multiobjective genetic algorithm for hardware-software cosynthesis of distributed embedded systems," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol.17, no.10, pp.920,935, Oct 1998

[8] Gruian, F.; Kuchcinski, K., "LEneS: task scheduling for low-energy systems using variable supply voltage processors," *Design Automation Conference, 2001. Proceedings of the ASP-DAC 2001. Asia and South Pacific*, pp. 449,455, 2001.

[9] Bambha, N.K.; Bhattacharyya, S.S.; Teich, J.; Zitzler, E., "Hybrid global/local search strategies for dynamic voltage scaling in embedded multiprocessors," *Hardware/Software Codesign, 2001. CODES 2001. Proceedings of the Ninth International Symposium on*, pp.243,248, 2001.

[10] Chandrakasan, A.P.; Brodersen, R.W., "Minimizing power consumption in digital CMOS circuits," *Proceedings of the IEEE*, vol.83, no.4, pp.498,523, Apr 1995.

[11] Sangyoung Park; Jaehyun Park; Donghwa Shin; Yanzhi Wang; Qing Xie; Pedram, M.; Naehyuck Chang, "Accurate Modeling of the Delay and Energy Overhead of Dynamic Voltage and Frequency Scaling in Modern Microprocessors," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol.32, no.5, pp.695,708, May 2013.

[12] S. Stuijk, M. C. W. Geilen, and T. Basten, SDF3: SDF For Free, in *Proceedings of the 6th International Conference Application of Concurrency to System Design*, Jun. 2006, pp. 276 278.

[13] Hiroyasu, T.; Nakayama, S.; Miki, M., "Comparison study of SPEA2+, SPEA2, and NSGA-II in diesel engine emissions and fuel economy problem," *Evolutionary Computation, 2005. The 2005 IEEE Congress on*, vol.1, no., pp.236,242 Vol.1, 5-5 Sept. 2005.

[14] W. M. Spears and V. Anand, A study of crossover operators in genetic programming, in *Proc. 6th Int. Symp. Methodologies for Intelligent Systems (ISMIS91)*, Z. W. Ras and M. Ze- mankova, Eds. Berlin, Germany: Springer-Verlag, 1991, pp. 409418.

[15] S. Cahon, N. Melab and E-G. Talbi, "ParadisEO: A Framework for the Reusable Design of Parallel and Distributed Metaheuristics", *Journal of Heuristics*, vol. 10(3), pp.357-380, May 2004.