An Emprical Study of Energy Consumption in Distributed Simulations*

Richard Fujimoto
School of Computational Science and Engineering
Georgia Institute of Technology
Atlanta, Georgia USA

Aradhya Biswas
Department of Computer Science and Engineering
Indian Institute of Technology Hyderabad
Yeddumailaram, Telangana, India

Abstract—**Power and energy consumption are important concerns in the design of high performance and mobile computing systems, but have not been widely considered in the design of parallel and distributed simulations. The importance of these factors is discussed and metrics for power and energy overhead in parallel and distributed simulations are proposed. Factors affecting the energy consumed by synchronization algorithms and software architectures are examined. An experimental study is presented examining energy consumption of the well-known Chandy/Misra/Bryant algorithm executing on a peer-to-peer mobile computing platform and compared with a centralized client-server approach using the YAWNS synchronization algorithm. Initial results concerning queueing network simulations are also presented. The results of this study suggest that existing distributed simulation algorithms require a significant amount of additional energy compared to a sequential execution. Further, different synchronization algorithms can yield different energy consumption behaviors.**

Keywords—parallel discrete event simulation; distributed simulation; power aware computing

I. Introduction

Power consumption has become a major concern for many parallel and mobile computing applications. The need to reduce energy use is clear in mobile and embedded computing where reductions result in increased battery life or enable the use of smaller batteries thereby reducing the size and weight of devices. In high-end computing energy consumption is a dominant cost associated with operating large data centers and supercomputers, and a substantial amount of effort has gone into developing techniques to mitigate this expense. Power consumption has become the key factor preventing substantial further improvements in clock speed and now limits computer performance. It has been cited as a major obstacle to creating supercomputers yielding exascale performance. Despite the importance of power and energy in computation today, very little attention to date has focused on understanding and developing techniques to minimize power and energy consumption in parallel and distributed simulations.

Energy is the capacity of a system to perform work. It is typically measured in units called joules where one joule is the work performed by an electrical circuit to move a charge of one coulomb through an electrical potential difference of one volt. Power is the amount of energy consumed per unit time with one watt of power defined as the expenditure of one joule of energy per second.

Minimizing energy usage and power consumption are not the same thing [1]. For example, decreasing the clock rate of the processor can lead to less power consumption. However, this will usually lead to longer execution times and an increase in the total amount of energy needed to complete a given computation. Battery operated devices are *energy-constrained* systems because they operate with a finite amount of available energy; thus a design goal might be to minimize the amount of energy utilized by the computation as a whole, subject to certain execution time constraints. On the other hand, in *power-constrained* systems such as supercomputers and data centers the amount of available energy is effectively unlimited, but a design goal may be to minimize the amount of time required to complete the computation given a certain maximum level of power consumption, or to minimize power consumption, subject to certain execution time constraints.

Power-aware and *energy-aware* systems are those where power or energy consumption is a principal design consideration. For example, power-aware systems may utilize techniques to change the system's behavior based on the amount of power being consumed. Energy-aware systems may modify the operation of the system based the amount of energy remaining in batteries.

It should be noted that minimizing execution time does not necessarily result in minimal energy consumption. Energy consumption is affected by many factors, e.g., the operation of the memory system, the number and complexity of computations performed by arithmetic circuits, and importantly, the amount of inter-processor communication that is required. A parallel or distributed computation that executes in a shorter amount of time may consume both more energy and more power if more communications are required.

Power- and energy-aware computing is increasing in importance for parallel and distributed simulation systems and applications. The main contribution of this paper is to highlight the increasing importance of power and energy consumption in parallel and distributed simulations. We propose performance metrics to quantitatively assess energy consumption in parallel and distributed simulations and report initial empirical measurements of the energy consumed by concervative synchronization algorithms.

* This research was supported by AFOSR grant FA9550-13-1-0100.

The next section describes a motivating application to highlight the relevance of energy consumption in distributed simulation. This is followed by a discussion of related work in this area by briefly surveying power- and energy-aware computing. Metrics for measuring energy and power consumption in distributed simulations are then proposed. The two distributed simulation systems examined in this study – a peer-to-peer system using the Chandy/Misra/Bryant algorithm and a client-server architecture using the YAWNS algorithm are then described. The experimental configuration and benchmark programs used in this study are presented, followed by a discussion of the power measurement methodology that is used. Experimental results are then presented and discussed, followed by concluding remarks and discussion of areas requiring further investigation.

II. A Motivating Application

Embedded mobile distributed simulations can be used to create adaptive sensor networks to monitor dynamically changing physical systems. For example, consider a collection of small, battery-operated unmanned aerial vehicles (UAVs) tasked with monitoring a physical system, e.g., tracking a set of vehicles moving throughout a city, monitoring the spread of a forest fire, or assessing the dispersion of a hazardous chemical plume following an accident (see Figure 1). Assume each UAV is equipped with sensors, an on-board computer, and wireless communications. Each UAV may initially be assigned to monitor a certain geographical area. It collects information concerning the state of the physical system in its immediate vicinity. Collectively the team of UAVs may then execute a distributed simulation to project the future state of the system, e.g., to project the location of the fire some time into the future in order to determine how best to relocate the UAVs in order to continue monitoring its spread. In some cases more UAVs may be assigned to monitor regions of particular interest, e.g., areas with higher traffic congestion, a larger density of fires, or higher concentrations of chemicals, leaving fewer UAVs to monitor areas projected to be less important to the monitoring activity. One can envision other similar surveillance applications involving teams of people carrying handheld devices or autonomous battery-powered ground vehicles.

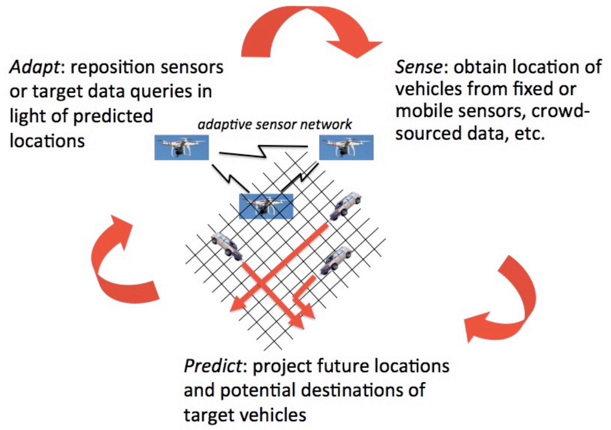
Adapt: reposition sensors or target data queries in light of predicted locations
Sense: obtain location of vehicles from fixed or mobile sensors, crowd-sourced data, etc.
adaptive sensor network
Predict: project future locations and potential destinations of target vehicles

Figure 1. Notional Diagram of a mobile data-driven distributed simulation system for monitoring traffic.

Adaptive sensor networks such as these could utilize centralized computing capabilities where sensors report information back to a command center where predictive simulations could be executed and the network reconfigured accordingly. Placing the simulations within the sensor network itself offers several advantages. First, it reduces or eliminates reliance on connectivity to the central command center, mitigating a potential point of failure. Further, a distributed implementation enables greater scalability than the centralized approach; one can envision many teams of UAVs that collaborate to monitor larger scale systems than would otherwise be possible. In certain applications embedding the simulation within the physical system itself enables faster response time for latency-critical applications.

Systems such as these are referred to as dynamic data-driven application systems (DDDAS) [2]. DDDAS involves incorporating live data from instrumented systems into executing applications in order to optimize the system and/or steer the measurement process. The DDDAS paradigm involves repeatedly executing a processing cycle of (1) sense, (2) predict future system states, and (3) adapt or reconfigure the system to optimize the physical system or to continue the monitoring process. DDDAS has been studied in a variety of applications. Our focus here is concerned with using embedded distributed simulation to implement the second step of the DDDAS processing cycle. It is clear that for these situations, energy consumption is an important concern when the simulation operates within battery-operated mobile devices. Exploitation of the DDDAS paradigm in UAVs is an active field of study and are discussed in (for example) [3, 4].

III. Related Work

There is a substantial literature in power- and energy-aware computing systems, and a variety of techniques that may be employed. Dynamic voltage and frequency scaling (DVFS) is concerned with altering the voltage and/or clock speed of the processor by taking into consideration energy and performance constraints [5-7]. Dynamic power consumption in CMOS circuits is proportional to FV^2 where F is the frequency at which the circuit is clocked, and V is the power supply voltage. Several commercial microprocessors support modification of the processor's frequency and voltage to trade off power consumption and performance. Scheduling algorithms for embedded systems have been designed to balance energy saving with meeting real-time deadlines, e.g., see [8-10]. Processors also commonly provide different modes of operation that utilize different amounts of power. Some work examines the utilization of these modes of operation, sometimes in conjunction with DVFS [11-13]. Other research examines issues such as predictive modeling of energy and power [14-16]; and embedded systems evaluations [17-20].

To date, work in power and energy aware computing has largely focused on low-level aspects of the computing system. Existing work focuses on effectively utilizing specific hardware capabilities, the development of operating systems and compilers, and communication protocols (e.g., routing algorithms in sensor and ad hoc networks) to reduce energy usage.

Only a modest amount of work to date has addressed energy and power consumption in parallel and distributed simulations. Some work has focused on characterizing power consumption for scientific computing applications [21-24]. One early effort examined power consumption for disseminating state information in distributed virtual environments, highlighting dead-reckoning algorithms and tradeoffs between state consistency and power consumption [25]. More recent work examined power consumption related to the implementation of data distributed management (DDM) services defined in the High Level Architecture [26]. To our knowledge, no prior work has examined power and energy consumption of synchronization algorithms for parallel and distributed simulations, the primary focus of the work described here.

IV. Energy and Power Metrics

In general, the three key metrics of greatest interest are the amount of energy, power, and time required to complete a computation. These metrics may be traded off against each other. For example, as discussed earlier one can trivially reduce power consumption by reducing clock frequency at the expense of increased execution time. In high performance computing contexts both execution time and power consumption are of interest, and balanced based on constraints such as a maximum level (ceiling) of power consumption. Similarly, in real-time applications both energy consumption and maximum execution time are important for best use of system resources while still meeting real-time constraints. For this reason the product of energy (or power) and execution time, referred to as the energy-delay product, is sometimes used as a metric that simultaneously considers both energy consumption and execution time [27].

A central concern here is the amount of additional energy consumed by the parallel/distributed execution that takes into account parallel/distributed computing overheads such as interprocessor message communication and synchronization. For this purpose we define a metric termed the *energy overhead*. Energy overhead refers to the amount of *additional* energy that is expended in the execution of a particular implementation of a parallel or distributed simulation on some hardware configuration relative to an energy-efficient sequential execution of the same computation.

This definition is motivated by the traditional definition of speedup. Like the speedup definition, "performance" is defined relative to an efficient sequential implementation. Also like speedup, this definition is meant to encourage the development of approaches to minimizing energy consumption recognizing that a baseline amount of energy must necessarily be consumed by the computation, just as speedup recognizes that a certain amount of time is required to complete the computation on a sequential machine. Energy overhead highlights the cost of the parallel/distributed implementation in terms of energy consumption.

Just as speedup may be determined using *strong* or *weak scaling*, similar methodologies apply in measuring energy overhead. In strong scaling the speedup is computed by comparing the execution time of a fixed sized sequential computation with a parallel implementation of the same computation distributed across a parallel processor. In this light *strongly scaled energy overhead* is computed as $E_P(N) - E_S$ where E_S is the energy consumed by a sequential implementation of the computation, and $E_P(N)$ is the energy consumed by a parallel/distributed implementation of the same computation distributed over N processors.

Alternatively, speedup computed using weak scaling involves scaling the size of the computation in proportion with the number of processors. Here, *weakly scaled energy overhead* is defined as $E_{P'}(N) - E_{S'}$ where $E_{S'}$ is the energy consumed by a computation C on a sequential machine and $E_{P'}(N)$ is the energy consumed by the same computation C executing on a single processor of the parallel/distributed machine with N processors and where the entire computation executed on the parallel/distributed machine is of size $C*N$. Weakly scaled energy overhead provides insight into the amount of additional energy consumed by the parallel or distributed computation as it is scaled to larger sizes in proportion to the size of the parallel/distributed computer.

The energy overhead is impacted by several factors. It clearly depends on the hardware configuration, including consideration of memory and communications circuits, and system software on which the parallel/distributed simulation executes. Energy consumption depends on any energy-saving techniques such as DVFS that are used. Our particular concern is the overhead associated with the synchronization algorithm. As will be discussed next, energy consumption depends on the software architecture used for the implementation.

The above discussion focused on energy overhead. Similar metrics for *power* overhead can also be defined. In this context, the power overhead refers to the additional amount of power required to execute the parallel/distributed simulation relative the sequential simulation.

V. Distributed Simulation Systems

In this study we compare two distributed simulation middleware approaches using conservative synchronization algorithms. These two approaches utilize a peer-to-peer and a client-server architecture, respectively. The context in which we envision these architectures to be deployed might be an embedded DDDAS application where the distributed simulation executes on a power-constrained mobile computing platform, possibly connected via wireless links to a local server. This architecture places the simulations in close physical proximity to online sources of data.

The "classic" approach to implementing a parallel discrete event simulation (PDES) program is to use a *peer-to-peer architecture* where each processor or node has approximately the same computational capabilities as other nodes. The logical processes (LPs) making up the PDES program are mapped to different computation nodes using a mapping algorithm or heuristic. LPs communicate directly with other LPs by sending messages to the appropriate nodes. Early work in PDES focused almost exclusively on this approach, and to this day, this is often considered to be the "default" approach to implementing a parallel or distributed simulation. It is depicted in Figure 2(a) below.

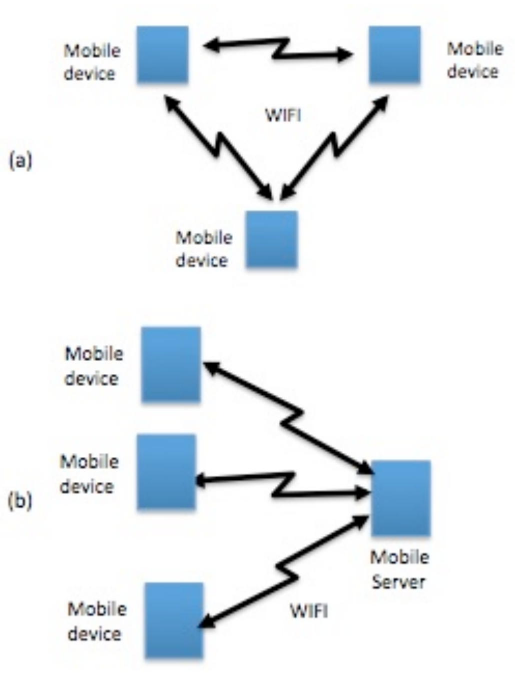

Figure 2. Peer-to-peer and client-server approaches.

In the peer-to-peer architecture a distributed algorithm is typically used to implement synchronization. Asynchronous algorithms use direct peer-to-peer communications between LPs (or processors). The Chandy/Misra/Bryant (CMB) algorithm is perhaps the most well known example of this approach [28, 29]. The principal source of energy overhead for this algorithm results from transmitting null messages between processors.

A second approach to distributed simulation is the client/server architecture, as shown in Figure 2(b). Logical processes execute within client processors while the simulation engine executes within the server. Here, in the context of the applications described earlier, we envision a mobile server that might reside in a special device, e.g., a larger gasoline-powered UAV. In general, however, the server might utilize the same mobile processor as client nodes in which case the distinction between clients and servers is largely logical, or the server might utilize a different, likely more powerful, machine.

Clients only communicate with the server; direct client-to-client communications are not allowed. Two key functions performed by the server include forwarding messages sent between LPs residing in different clients and synchronization among the LPs/clients. This architecture is sometime used in federated distributed simulations, e.g., those based on the High Level Architecture standard, where RTI services are for the most part implemented within the server.

A natural, straightforward approach to implementing synchronization in client-server architectures are synchronous algorithms that utilize global synchronization points. The computation executes through a sequence of epochs where each involves determining those events that can be executed in this epoch with timestamps that are guaranteed to be smaller than any event that might later be received. Well known synchronous algorithms include YAWNS [30] and Bounded Lag [31]. A principal source of energy overhead for this algorithm lies in the barrier synchronization mechanism and LBTS computation that is required. Each epoch includes a barrier and computation of the lower-bound-on-timestamp (LBTS) value indicating the minimum timestamp of any event it might be generated in the future. More precisely, each client processor computes the smallest time stamp for any new message it might produce in the absence of receiving any additional messages as $T_i + L$ whee T_i is the timestamp of the next unprocessed local event within the processor and L is the looakhead for the processor. LBTS is defined as the minimum among all of these values produced by the different processors. All events with timestamp less than LBTS are safe to process.

VI. Experimental Configuration and Benchmark Application

The experimental configuration is intended to mimic an embedded distributed simulation application where the distributed simulation executes within a set of mobile processors. In the peer-to-peer system the mobiles make up the entire hardware platform. In the client-server architecture we posit the server resides in a location where a power source is readily available so energy use within the server is of secondary importance.

Experiments were performed to compare the CMB (peer-to-peer) and YAWNS (client-server) algorithms. A LG Nexus 5 cellular phone with a quadcore Qualcomm MSM8974 Snapdragon 800 processor, 2 GB memory, and 16 GB storage was used as the mobile computing platform. While the systems we envision may not necessarily use cellular phones as their compute engine, they most likely will utilize the same mobile processors that are used in cellular phones. The phone runs the Android version 5.0.1 (Android Lollipop) operating system and was used in the peer-to-peer experiments. The same phone was used as the client in the client server architecture. A laptop was used as the server for the latter architecture. Hardware-based techniques to reduce power consumption such as voltage or frequency scaling were not used in these experiments.

All inter-processor communications utilizes wireless links. In both cases the device's 802.11n WiFi network interface was used for communications between processors. A private wireless network was established among the devices to avoid interference resulting from Internet traffic. The cellular network capability of the phone is not used in these experiments.

The energy and power consumption data are derived from direct measurement of the Android device. Specifically, the value of the instantaneous power being consumed by the device is calculated by multiplying the instantaneous battery current given by the constant integer BATTERY_PROPERTY_CURRENT_NOW and the current battery voltage level given by the constant string EXTRA_VOLTAGE. Both of these appear in the "BatteryManager" class of the "android.os" API. The instantaneous power consumption so obtained is then used to calculate the energy consumption over time.

A. *Energy Used By Synchronization Algorithms*

An initial set of experiments were conducted to measure the amount of power used by the synchronization algorithm. This was accomplished by creating benchmark programs where each LP only processed local events, and did not exchange events with other processors. This ensures that interprocessor communication is only utilized by the synchronization algorithm rather than passing event messages. In these experiments each LP is initialized with some number of local events, and processing each event causes one new locally scheduled event to be scheduled with a fixed time stamp increment. A set of experiments were then performed using the CMB and YAWNS implementations as lookahead was varied. Because no events are scheduled between processors, the lookahead could be set to arbitrary values without concern of violating lookahead constraints. In these experiments the simulation benchmark program is the same across all lookahead values and both synchronization algorithms, enabling fair comparisons.

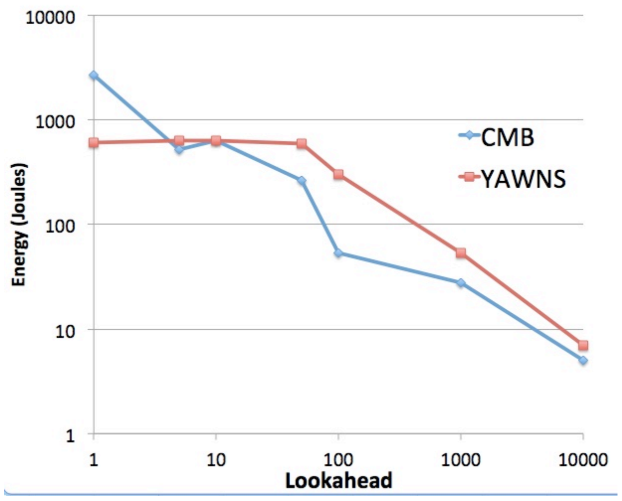

Figure 3. Energy consumed for Chandy/Misra/Bryant and YAWNS synchronization algorithms as lookahead is varied.

The amount of energy consumed by the benchmark programs for different lookahead values are shown in Figure 3. In this figure both lookahead and energy are plotted on logarithmic scales.

It is seen that lookahead can have a dramatic effect on the amount of power consumed by the synchronization algorithm – two orders of magnitude across the lookahead values used in these experiments.

The CMB algorithm yielded energy consumption that steadily decreased as the lookahead was increased. For very small lookahead values CMB is prone to a phenomena called lookahead creep where null messages must be sent among the processors to, in effect, enable them to advance by an amount of simulation time equal to the lookahead. To a first-order approximation, doubling the lookahead value approximately doubles the amount of time advance that can be gained with each "round" of null messages. Assuming the primary cause of energy utilization is the time to send null messages, the data shown in Figure 2 is consistent with this observation where it is seen a steady decline in energy consumption results as the lookahead value is increased.

The YAWNS experiments yielded a decidely different behavior. Here, the energy consumptions remains at a relatively constant level for small to moderate lookahead values. However, energy consumption then steadily decreses with lookahead increases at relatively high lookahead values. In contrast to CMB, YAWNS is not prone to the lookahead creep problem in the sense that the algorithm exploits knowledge of the timestamp of the next unprocessed event to advance simulation time. Consider the case where the lookahead is very small, say 1, and the average time between events is 10 units of simulation time. In accordance with time creep, CMB must advance each LP by increments of 1 with each round of null messages to advance LPs to the point where they can process the next (non-null) event. On the other hand, YAWNS will immediately advance the LP to the time of the next event. If the lookahead is small, YAWNS will, again to a first-order estimate, advance LBTS to the timestamp of the next unprocessed simulation event. Therefore the energy required for synchronization is in proportion to the number of events, independent of the lookahead value, explaining why energy consumption remains flat for small lookahead values. This remains true until the lookahead becomes large. With large lookahead values, many events can be processed in each epoch of YAWNS. Roughly speaking, doubling the lookahead value approximately doubles the number of events that can be processed within each epoch. Thus, for large lookahead values, the energy overhead steadily declines as the lookahead increases. The data in Figure 2 is consistent with this explaination.

Overall, it can be seen that YAWNS and CMB use roughly comparable amounts of energy in the experiments with large lookahead values. However, these measuements also suggest CMB expends considerably more energy at very low lookahead values due to the lookahead creep problem.

B. *Queueing Network Simulations*

A second benchmark program used in this study is a simulation of a closed queueing network with *J* jobs circulating among the nodes of the network. The queueing network is configured as a three-dimensional toroid topology. Each processor is assigned one two-dimensional plane of the toroid. Once a job receives service it is routed to a randomly selected neighboring node, with each neighbor equally likely to be selected. Each node of the network contains a single server with service time drawn from an exponential distribution plus a constant value *L*. Jobs arriving at each network node are placed into a single queue, and are served in first-come-first-serve order. The minimum service time *L* is used as a control variable to facilitate experimentation with increased lookahead values. In these experiments the lookahead is enhanced by pre-sampling the random number generator to produce the service time of the next job to be processed by the server; if the pre-sampled value is *P*, then the time stamp of the next message generated by the LP must be at least *L*+*P* units of simulation time into the future [32]. Further, the simulation is optimized to exploit the fact that the queue uses a FCFS queueing discipline, resulting in increased lookahead in proportion to the queueing delay. The benchmark program is written in C.

Figure 4 shows the energy consumed by the three simulations for queueing networks of size 4 (2x2), 49 (7x7), 484 (22x22), and 1024 (32x32) nodes executing on each processor. The total number of events processed by the simulators was kept constant across all of these runs, i.e., the total amount of simulation computation remained the same across the runs. This figure shows energy consumption data; power consumption data demonstrated similar trends to that shown in the figure.

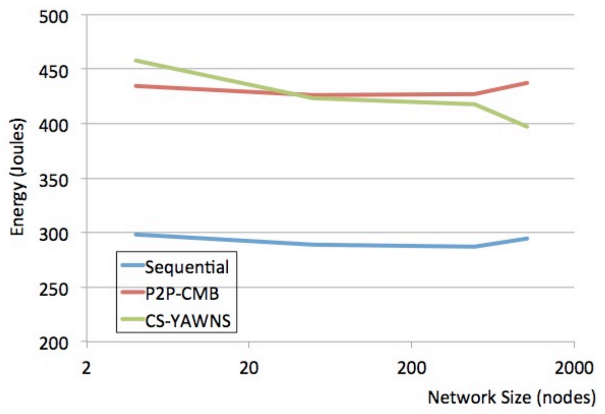

Figure. 4. Energy consumption for 2x2, 7x7, 22x22, and 32x32 queueing networks. Note network size is plotted on a logarithmic scale.

It can be seen that the energy consumed by the sequential and P2P-CMB simulations remains about constant as network size is increased, but there is a modest reduction of approximately 13% in energy consumed by the CS-YAWNS simulation for the largest network compared to the smallest. We believe this is due to a much smaller number of synchronization messages in CS-YAWNS in the larger sized queueing networks. YAWNS operates on a time window scheme where all events in the current time window can be safely processed without concern for events later arriving with a smaller timestamp. For these experiments the lookahead, i.e., the minimum timestamp increment, remains the same across all runs. Therefore, there will be more events within a single time window that may be processed before the next global synchronization. Since the total number of events in the computation remains the same, this results in fewer global synchronization points. The number of synchronization messages in YAWNS was reduced in approximately the same proportion as the size of the network (a factor of 256) across these experiments.

Varying the size of the network changes the amount of computation performed between communications. Larger networks will have more simulation computations to complete between successive interprocessor message communications, likely accounting for the difference shown in this figure.

The energy overhead resulting from the distributed execution of the simulation program relative the sequential implementation using these two synchronization algorithms and architectures is shown in Figure 5. These data are derived by subtracting the energy consumed by the sequential implementation from the distributed execution for each network size, and plotting the resulting value as a percentage of the energy expended in the sequential execution. As can be seen, the distributed simulations expend from 35% to 54% more energy than the sequential simulation executing the same number of events. While the P2P implementation of CMB remains approximately the same percentage of energy overhead for the different sized networks, there is more variability in the client-server YAWNS implementation.

VII. DISCUSSION

One can observe a few trends that emerge across these experiments. First, these data highlight the energy cost resulting from the distributed execution of a simulation program is significant. Without more detailed measurements of the architecture itself, one cannot definitively pinpoint all of the causes of this increased energy usage, however it seems clear that interprocessor communication for event passing and synchronization is most likely a principal factor. These measurements indicate that the energy overhead is significant for these conservative synchronization algorithms.

Second, we observe that different synchronization algorithms and architectures exhibit different behaviors with respect to energy consumption. It is clear that different synchronization algorithms will exhibit different message passing behaviors, so in this sense it is not surprising that they yield different energy consumption characteristics. These data indicate that these differences can be significant, and can lead to observable differences in energy consumption. Thus, for applications where energy is a critical factor, e.g., for distributed simulations executing on mobile devices, some care should be taken with respect to the choice of synchronization algorithm in order to maximize battery life.

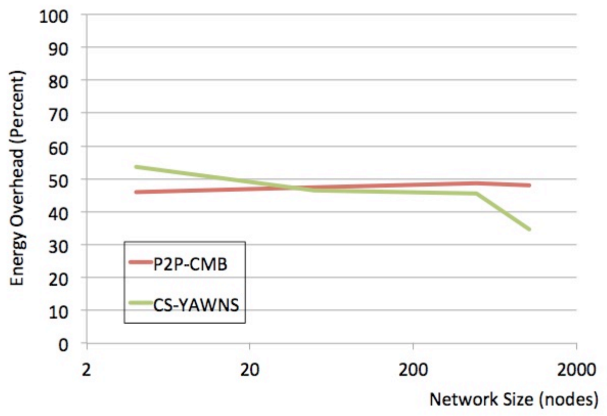

Figure 5. Energy overhead of CMB and YAWNS for different sized queueing networks as a percentage of the energy expended by the sequential execution.

Third, aspects of the distributed simulation application such as lookahead that impact the behavior of the synchronization algorithm may have a significant impact on the energy efficiency of the distributed simulation. Further investigation is required to examine the impact of aspects such as lookahead on energy efficiency and to gain a deep understanding of this relationship.

When comparing the energy consumption of these two, very different, synchronization algorithms, two observations are apparent. First, to a first order approximation, the overall, average energy overhead observed for these synchronization algorithms is significant, and to some extent, relatively similar. One the other hand, the two algorithms exhibit different energy characteristics as parameters of the simulation such as the number of LPs and lookahead change. One must be cautionary in that these observations are based on a very limited amount of experimental data. Nevertheless, these results suggest that further exploration of the relationship between the synchronization algorithm and energy and power consumption is warranted.

VIII. Conclusions and Future Work

We have argued that power and energy consumption are areas of increasing concern for parallel and distributed simulation systems. In contexts such as embedded and mobile systems and some high performance computing applications we believe these aspects are of sufficient importance to merit explicit consideration in the design of the system.

Metrics are proposed focusing on the energy and power efficiency of parallel and distributed simulations relative to a sequential simulation. The metrics focused on developing measures that are consistent and complementary to standard metrics used for some time, specifically speedup under strong and weak scaling assumptions.

The empirical work presented here represents an initial evaluation of the energy and power consumed for parallel and distributed simulations, focusing on the synchronization algorithm that is used. Experimental measurements of operational distributed simulation systems demonstrate that distributed execution incurs a significant overhead in energy consumption to execute the simulation. Clearly the amount of this overhead will depend on the application, but queueing networks, a standard benchmark used for parallel and distributed simulation, of various sizes all demonstrate significant energy overhead. Architectural choices and the synchronization algorithm can lead to different results concerning the amount of energy and power that is consumed. In these experiments the client-server YAWNS implementation and the Chandy/Misra/Bryant algorithm executing on a peer-to-peer architecture using WiFi communications yielded comparable energy overheads overall, though the two approaches exhibited different detailed behaviors for different network sizes.

Power- and energy- consumption of parallel and distributed simulations represents a new area of research that has many unanswered questions. Deep understandings of the power and energy consumed by distributed simulations, and those aspects that are different relative to other types of computing applications do not yet exist. Accurate, predictive models of energy consumption of parallel and distributed simulations taking into account the machine architecture as well as the behavior of the simulation application are needed. A comprehensive examination of alternate conservative synchronization algorithms and their execution on different distributed system architectures across a wide variety of parameter settings is required. Similar evaluations and comparisons with optimistic synchronization techniques are needed. Extensive analyses for real world applications are needed both for embedded and mobile simulations as well as high performance computing systems.

Beyond analysis studies, the development and evaluation of techniques to reduce energy- and power- consumption of distributed simulations is an unexplored area of research. We refer to techniques that take into consideration power and energy consumption in the design and operation of the system energy- and power-aware distributed simulations. One can envision dynamically changing the behavior of the distributed simulation based on the energy available in batteries, or based on the power being consumed on a high performance computing platform. Distributed simulations executing in a real-time context such as those that arise in DDDAS impose time constraints to completing the computation. Moreover, we believe power- and energy-aware parallel and distributed simulation represents a rich area of future research for the field with many unsolved problems.

References

[1] O. S. Unsal, "System-Level Power-Aware Computing In Complex Real-Time and Multimedia Systems," Doctor of Philosophy Doctoral Dissertation, Department of Electrical and Computer Engineering, University of Massachusetts, Amherst, 2008.

[2] F. Darema, "Dynamic Data Driven Applications Systems: A New Paradigm for Application Simulations and Measurements," presented at the International Conference on Computational Science, Kraków, Poland, 2004.

[3] F. Kamrani and R. Ayani, "Using On-Line Simulation for Adaptive Path Planning of UAVs," in *Proceedings of the 11th IEEE International Symposium on Distributed Simulation and Real-Time Applications*, 2007, pp. 167-174.

[4] G. R. Madey, M. B. Blake, C. Poellabauer, H. Lu, R. R. McCune, and Y. Wei, "Applying DDDAS Principles to Command, Control and Mission Planning for UAV Swarms," in *Proceedings of the International Conference on Compuational Science*, 2012.

[5] R. Ge, X. Feng, and K. W. Cameron, "Performance-constrained Distributed DVS Scheduling for Scientific Applications on Power-aware Clusters," presented at the Proceedings of the 2005 ACM/IEEE conference on Supercomputing, Washington, DC, USA, 2005.

[6] V. W. Freeh, D. K. Lowenthal, F. Pan, N. Kappiah, R. Springer, B. L. Rountree*, et al.*, "Analyzing the Energy-Time Trade-Off in High-Performance Computing Applications," *IEEE Trans. Parallel Distrib. Syst.,* vol. 18, pp. 835--848, June 2007.

[7] S. Hua and G. Qu, "Approaching the Maximum Energy Saving on Embedded Systems with Multiple Voltages," presented at the IEEE/ACM International Conference on Computer-Aided Design, 2003.

[8] S. Saewong and R. Rajkumar, "Practical voltage- scaling for fixed-priority rt-systems," presented at the IEEE Real- Time and Embedded Technology and Applications Symposium, 2003.

[9] G. Quan and X. Hu, "Energy efficient fixed- priority scheduling for real-time systems on variable voltage processors," presented at the Design Automation Conference, 2001.

[10] K.-M. Cho, C.-H. Liang, J.-Y. Huang, and C.-S. Yang, "Design and implementation of a general purpose power-saving scheduling algorithm for embedded systems," presented at the IEEE International Conference

on Signal Processing, Communications and Computing, Xi'an, China, 2011.

[11] A. Hoeller, L. Wanner, and A. Fröhlich, "A hierarchical approach for power management on mobile embedded systems," in *From Model-Driven Design to Resource Management for Distributed Embedded Systems*, ed, 2006, pp. 265–274.

[12] K. Bhatti, C. Belleudy, and M. Auguin, "Power management in real time embedded systems through online and adaptive interplay of DPM and DVFS policies," presented at the International Conference on Embedded and Ubiquitous Computing, Hong Kong, China, 2010.

[13] L. Niu and G. Quan, "Reducing both dynamic and leakage energy consumption for hard real- time systems," presented at the international conference on Compilers, architecture, and synthesis for embedded systems, 2004.

[14] K. Czechowski and R. Vuduc, "A Theoretical Framework for Algorithm-Architecture Co-design," presented at the Parallel Distributed Processing (IPDPS), 2013 IEEE 27th International Symposium on, 2013.

[15] S. Williams, A. Waterman, and D. Patterson, "Roofline: an insightful visual performance model for multicore architectures," *Commun. ACM,* vol. 52, pp. 65--76, April 2009.

[16] S. W. Keckler, W. J. Dally, B. Khailany, M. Garland, and D. Glasco, "GPUs and the Future of Parallel Computing," *Micro, IEEE,* vol. 31, pp. 7 -17, sept.-oct. 2011.

[17] I. Grasso, P. Radojkovic, N. Rajovic, I. Gelado, and A. Ramirez, "Energy Efficient HPC on Embedded SoCs: Optimization Techniques for Mali GPU," presented at the Parallel and Distributed Processing Symposium, 2014 IEEE 28th International, 2014.

[18] N. Rajovic, P. M. Carpenter, I. Gelado, N. Puzovic, A. Ramirez, and M. Valero, "Supercomputing with Commodity CPUs: Are Mobile SoCs Ready for HPC?," presented at the Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis, New York, NY, USA, 2013.

[19] N. Rajovic, A. Rico, J. Vipond, I. Gelado, N. Puzovic, and A. Ramirez, "Experiences with mobile processors for energy efficient HPC," presented at the Design, Automation Test in Europe Conference Exhibition (DATE), 2013, 2013.

[20] L. Stanisic, B. Videau, J. Cronsioe, A. Degomme, V. Marangozova-Martin, A. Legrand*, et al.*, "Performance analysis of HPC applications on low-power embedded platforms," presented at the Design, Automation Test in Europe Conference Exhibition (DATE), 2013, 2013.

[21] J. Dongarra, H. Ltaief, P. Luszczek, and V. M. Weaver, "Energy Footprint of Advanced Dense Numerical Linear Algebra using Tile Algorithms on Multicore Architecture," presented at the The 2nd International Conference on Cloud and Green Computing, 2012.

[22] X. Feng, R. Ge, and K. W. Cameron, "Power and Energy Profiling of Scientific Applications on Distributed Systems," presented at the Proceedings of the 19th IEEE International Parallel and Distributed Processing Symposium (IPDPS), 2005.

[23] R. Ge, X. Feng, S. Song, H.-C. Chang, D. Li, and K. W. Cameron, "PowerPack: Energy Profiling and Analysis of High-Performance Systems and Applications," *IEEE Transactions on Parallel and Distributed Systems (TPDS),* vol. 21, pp. 658-671, May 2010.

[24] H. Esmaeilzadeh, T. Cao, X. Yang, S. M. Blackburn, and K. S. McKinley, "Looking back and looking forward: power, performance, and upheaval," *Commun. ACM,* vol. 55, pp. 105--114, July 2012.

[25] W. Shi, K. S. Perumalla, and R. M. Fujimoto, "Power-aware State Dissemination in Mobile Distributed Virtual Environments," in *Workshop on Parallel and Distributed Simulation*, San Diego, 2003.

[26] S. Neal, G. Kanitkar, and R. M. Fujimoto, "Power Consumption of Data Distribution Management for On-Line Simulations," presented at the Principles of Advanced Discrete Simulation, Denver, Co., 2014.

[27] R. Gonzalez and M. Horowitz, "Energy Dissipation in General Purpose Microprocessors," *IEEE Journal of Solid-State Circuits,* vol. 31, pp. 1277-1284, September 1996.

[28] K. M. Chandy and J. Misra, "Distributed Simulation: A Case Study in Design and Verification of Distributed Programs," *IEEE Transactions on Software Engineering,* vol. SE-5, pp. 440-452, 1979.

[29] R. E. Bryant, "Simulation of Packet Communication Architecture Computer Systems," M.S. thesis, MIT-LCS-TR-188, Computer Science Laboratory, Massachusetts Institute of Technology, Cambridge, Massachusetts, 1977.

[30] D. M. Nicol, "The Cost of Conservative Synchronization in Parallel Discrete Event Simulations," *Journal of the Association for Computing Machinery,* vol. 40, pp. 304-333, June 1993.

[31] B. D. Lubachevsky, "Efficient Distributed Event-Driven Simulations of Multiple-Loop Networks," *Communications of the ACM,* vol. 32, pp. 111-123, 1989.

[32] D. M. Nicol, "Parallel Discrete-Event Simulation of FCFS Stochastic Queueing Networks," *SIGPLAN Notices,* vol. 23, pp. 124-137, 1988