SimGate: Full-System, Cycle-Close Simulation of the Stargate Sensor Network Intermediate Node

Ye Wen, Selim Gurun, Navraj Chohan, Rich Wolski, and Chandra Krintz
Department of Computer Science
University of California, Santa Barbara

***Abstract*— We present SimGate – a full-system simulator for the Stargate intermediate-level, resource-constrained, sensor network device. We empirically evaluate the accuracy and performance of the system in isolation as well as coupled with simulated Mica2 motes. Our system is functionally correct and achieves accurate cycle estimation (i.e. cycle-close). Moreover, the overhead of simulated execution is modest with respect to previously published work.**

I. Introduction

Sensor networks have emerged as a technology for transparently interconnecting our physical world with more powerful computational environments, and ultimately, global information systems. In a typical sensor network, computationally simple, low-power, sensor elements take physical readings and may perform minor processing of these readings before relaying them to more powerful computational devices. The need for non-intrusiveness influences sensor design toward small, inexpensive, low-power sensor implementations that can be deployed in large numbers. Because the sensor nodes are so resource constrained, complex tasks such as data storage and advanced processing are typically offloaded to more capable nodes.

Designing and investigating these systems, to date, has relied primarily on physical deployments and experimentation [1], [2], [3], [4], [5]. While the quality of the results from such efforts is invaluable, the requirement that researchers work directly with the physical systems imposes a substantial research impediment and financial cost. Alternatively, several systems [6], [7], [8], [9], [10], [11], provide simulation, which complement deployed experiments and offer flexible and controllable environment for sensor network development, analysis, and application prototyping. However, extant approaches to sensor network simulation do not support simulation of intermediate-level devices and the interactions between devices at different levels.

In this paper, we investigate SimGate – a full-system sensor network simulator. SimGate simulates the Intel Stargate device [12]. The Stargate device is intended to function as a general purpose processing, storage, and network gateway element in a sensor network deployment. Our goal is to provide both functional correctness and accurate cycle estimation. We refer to the latter as *cycle-close* [13]. Our system is unique in that is supports cycle-close, functional simulation of the entire Stargate device as opposed the processor [14] or power consumption [15], [16] alone. SimGate captures the behavior of most Stargate components including the processor, memory hierarchy, communications (serial and radio), and peripherals. As the result, SimGate boots and runs the Familiar Linux operating system and any program binary that executes over it, *without modification*.

Our system is also unique in that we use it to simulate the ensemble of both intermediate and base level sensor devices, i.e. Stargate and motes (e.g. Mica2). We do so by coupling SimGate with SimMote, a cycle-accurate full-system simulator of motes [17]. This interoperability reveals the potential of simulating a complete sensor network setting including both basic nodes, gateway nodes, and their interactivity.

We evaluate the accuracy of our system by comparing the simulated clock cycles to measured clock cycles using a range of stressmarks and community benchmarks. We also present results for similar experiments in which the SimGate and SimMote inter-operate via a serial interface (simulated in both). Finally, we examine a multi-device ensemble consisting of a SimGate node, a serially-connected SimMote, and a third SimMote that communicates only via simulated radio. We compare our simulated results to measurements that we gather from physical Stargate and Mica2 devices.

Our results indicate that our system is cycle-close – we are able to simulate the full Stargate system with a *maximum error* of 12.4% across the benchmark that we considered. Our simulator is also fast: we observe a $58X$ slowdown over real-time device execution. Moreover, when functional simulation is acceptable, simulation is $3X$ faster than cycle-close simulation. As a result, we believe this work demonstrates the potential of multi-device, sensor network simulation as a research-enabling technology.

In the next section, we overview the design and implementation of our simulator. In Section III, we describe our experimental setup and measurement methodology. We then detail the accuracy and performance of our system in Section IV. In Sections V and VI, we present related work and conclude with some observations and our plans for future work respectively.

II. SimGate Simulator

Simulation is an important tool for sensor network system and application development. The focus of most prior work in simulation has been on high-end devices [18], [19], [20], processor and power simulation [21], [14], [15], [16], or on the sensing devices themselves [6], [7], [8], [9], [10], [11]. However, to our knowledge, no extant approach to sensor

network simulation enables full-system simulation of a key sensor network component, the intermediate "gateway" node. Moreover, no simulation system facilitates co-simulation of different sensor network devices as part of an ensemble. The goal of our work is to investigate, implement, and evaluate such techniques.

To simulate intermediate nodes, we developed SimGate, which provides a cycle-close simulation of the Intel XScale processor [22]. Furthermore, SimGate emulates the complete functionality of the Stargate, such that it can boot and execute the Stargate software distribution (Linux kernel and programs) without any software modification. In addition, SimGate provides a unified debugging interface for program development.

A. SimGate Design and Implementation

The Stargate is a single-board, embedded system that consists of a 400MHz Intel XScale processor, an Intel SA1111 companion chip for I/O, Intel StrataFlash, SDRAM, PCMCIA/CF slots, and serial connector for a mote sensing device [12]. Currently, the Stargate does not have a mote-compatible RF radio. In situ, any Stargate-mote radio communication must be relayed by another mote, called the base station, that is physically attached to the Stargate expansion bus.

We simulate the following features of the Stargate device:

- ARM v5TE instruction set without Thumb support and with XScale DSP instructions
- XScale pipeline simulation, including the 32-entry TLBs, 128-entry BTB, 32KB caches and 8-entry fill/write buffers
- PXA255 processor, including MMU (co-processor), GPIO, interrupt controller, real time clock, OS timer, and memory controller
- Serial device (UART) that communicates with the attached mote
- SA1111 StrongARM companion chip
- 64MB SDRAM chip
- 32MB Intel StrataFlash chip
- Orinoco wireless LAN PC card including the PCMCIA interface

We are currently working to develop Stargate power models.

One expensive operation in our simulation system is memory access. Simulation of MMU components, including the TLB, BTB, I/D caches, and fill/write buffers, is complex but required for cycle-close simulation. However, such fine-grain simulation is not necessary for functional simulation. As such, users of our system can turn off the simulation of individual MMU components to improve simulation performance when cycle accuracy is not a primary concern. To further increase the address translation speed, we implement an address lookup cache (soft TLB) for both instruction and data addresses. This soft TLB increases functional simulation time by 10% on average.

To estimate cycle counts accurately, we simulate fine-grain XScale CPU pipeline process. The Intel XScale core employs a seven or eight stage, single-issue pipeline. There are three parallel pipeline stages that execute in parallel after the execution stage. As a result, multiplication and memory access can occur concurrently and the results can be written back to memory out of order. Since we were unable to obtain pipeline documentation from Intel, we revise and enhance a reference pipeline model from XTREM power simulator [16]. We run hand-coded benchmarks on a real device and use hardware performance monitors to estimate various pipeline parameters, such as cache and TLB miss penalties.

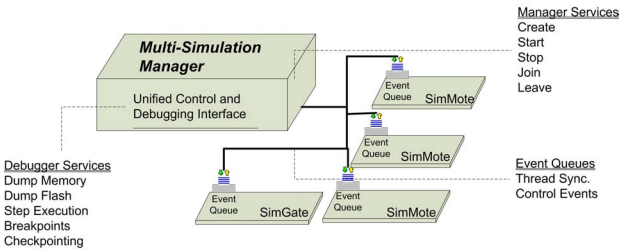

Fig. 1. Coupled Simulation System.

The peripheral and I/O devices that we simulate include the flash chip and the Orinoco PCMCIA wireless card. The flash chip is controlled by memory mapped I/O registers. The simulator sends and receives I/O commands and data through these registers. In the flash chip, a state machine controls the sequence of operations. We simulate both the interface and the internal state machine according to a Verilog model of the flash chip from Intel (`http://www.intel.com/design/flcomp/toolbrfs/298189.htm`). We estimate the cycle cost of flash operations using empirical data; we run flash benchmarks, collect timing data and use it to advance the simulation clock accordingly.

Since the 802.11b wireless is not commonly used in sensor network, we do not discuss its implementation detail in this paper. To communicate between the Stargate and sensor nodes, researchers attach a mote to the serial port of the Stargate and use the mote to interface to the radio network of other sensor nodes.

B. Coupling SimGate with Other Sensor Network Simulators

We next describe how we co-simulate Stargate and sensing devices, e.g. Crossbow Motes [23], [24]. Figure 1 depicts the system. We use SimMote for cycle-accurate, full-system mote (Mica2 and Mica2) simulation, which we developed in prior and ongoing work [17] on scalable sensor network simulation. To couple device simulators, we employ a multi-simulation manager. This manager is a multi-threaded system that controls the life cycle of constituent simulators, e.g., it provides simulator services that include create, start, stop, join and leave. To start a simulation, the manager creates a thread for each simulator and invokes the start routine in each. The start routine then initiates the OS boot process. A configuration file specifies which binary to boot for each simulator.

To achieve cycle-close, coordinated simulation of multiple devices, the proportion of the rates of execution of simulated devices must be held to be roughly the same as that for real devices. This coordination is critical for correct execution and communication (e.g., for a radio or serial connection).

To enable this coordination, we employ a simple, lock-step method that forces the clock within each simulator to synchronize periodically. This period is bounded by the mote-Stargate communication rate (57.6 KB/second). We use the one byte serial transmission time (128 mote cycles) as the period.

For communication, we distribute each packet to the receiving device simulator which assembles the packet using its local clock. Since the simulators execute in *lock-step*, the synchronization period is sufficient to ensure correct packet assembly (in a rate at 19.2KB/s).

The lock-step synchronization forces us to simulate the motes *as slow as* the Stargate. Since the machine on which we run our simulations is much faster than the real speed of the mote, we can simulate up to 6 times faster than real Mote execution. However, in an ensemble system of heterogeneous device simulators, the fastest machine simulated is the performance bottleneck. As such, we must slow the SimMotes to match SimGate speed.

C. Other Simulation Framework Features

To facilitate sensor network application development, monitoring, and understanding, our system includes functionality for both checkpointing and debugging. Our checkpointing mechanism within each simulator saves the current, full-system, simulation state including the snapshot of memory and flash file system. We provide mechanisms that enable storage and loading of such images to enable fast forwarding and continuation of an executing system.

To facilitate debugging, our system provides a unified debugging interface and dispatch within the multi-simulation manager that enables debugging of concurrently executing tasks. The manager dynamically dispatches debug commands to the individual simulators. Since each simulator runs on a separate thread, the debugger can attach to any of the simulator threads to control its execution flow and to watch the change in the execution state. The functions we support in the simulators include step execution, the dump of memory and flash, and watching of internal state and break points.

III. Experimental Methodology

To evaluate and analyze the performance and accuracy of SimGate, we performed a number of experiments with the SimGate alone and with SimGate-SimMote ensembles. To evaluate the latter, we implemented two scenarios: (1) A mote attached to a Stargate; and (2) A secondary mote communicating with the first via simulated radio.

Scenario (1) represents the use of the Stargate as a gateway, whereas Scenario (2) models a gateway and a mote. In the latter one, the gateway act as a packet forwarding engine to and from the sensor network. At present, our radio model discount the RF interference, however, we are currently working on robust and accurate radio models as part of future work.

A. Benchmarks

For stand-alone SimGate evaluation, we employed hand-coded "stressmarks" and benchmarks from the suites of both MiBench [26] and Mediabench [27]. Due to space constraints, we only include the results from Mediabench in this paper; our technical report version of this paper contains all of the data [28]. Table I describes the Stargate benchmarks. We execute all the benchmarks from the RAM drive.

To evaluate ensemble simulation, we employ open-source applications as well as hand-coded programs. We describe the applications in Table II. Column 3 shows the functional units of the motes that are heavily utilized during the execution of various benchmarks. In choosing benchmarks, we attempt to exercise the full device, and cover the major functions of a mote: communication, sensing, and logging.

Each ensemble benchmark has a *Short* and a *Long* form. The former are used for evaluating Scenario 1, whereas the latter are used for Scenario 2. Moreover, each of these applications (except the Multi benchmarks) takes the form of a remote procedure call (RPC). When the program on the Stargate sends a query to the mote, it blocks until the receiver completes the appropriate execution and returns. The Multi benchmark also tests concurrent computation by running parallel computations of ad-hoc positioning system (APS) [25] on both mote and Stargate. This test is useful to evaluate the performance of simulating coordinated computation on mote and Stargate.

B. Experimental Apparatus

We execute TinyOS v1.1 on the motes and a variation of Familiar Linux v0.5.1 on the Stargate. For the stand-alone Stargate applications, we measured the CPU cycles and instruction count using the XScale HPM counters. The HPM system can monitor three events concurrently.

We ran our simulators on a dedicated Linux (64 bit AMD Opteron@2.4GHz) machine. To measure the execution time of each benchmark, we modified the simulator. Each time the performance monitoring registers of the simulated machine are accessed, the simulator reads the real time from the host system, and computes and logs the delta (time since previous access).

IV. Results

We detail the accuracy of SimGate by comparing it to the Stargate in terms of clock cycles. In the first set of comparisons, we make 20 identical runs of each benchmark on both SimGate and Stargate and compare the average number of cycles required per benchmark.

Table III shows the result of this comparison in the following format. The first column shows the name of the benchmark, the second column (μ_{meas}) shows the average number of cycles measured on the Stargate hardware, the third column ($\mu_{simulated}$) presents the cycles reported by SimGate , and the fourth column shows the difference. In the fifth column, we report the error percentage ($|(\mu_{meas} - \mu_{simulated})/\mu_{meas}|$) which is the difference between the average of the measured cycle counts and the average of those generated by the simulator. We also compute the 95% confidence interval for the error percentage using a Student t distribution [29] with

Benchmark	Executables	Description
adpcm	adpcmdecode/adpcmencode	Adaptive differential pulse code modulation for audio coding
g721	g721decode/g721encode	CCITT voice compression
gsm	gsmencode/gsmdecode	European standard for speed coding
jpeg	jpegencode/jpegdecode	Lossy compression for still images

TABLE I

BENCHMARKS (FOR STARGATE PLATFORM).

Benchmark	Description	Functional Unit
Ping	Echoes network packet back to sender	Network interface
Sense	Processes a sensor read query	Analog/Digital converter
APS [25]	Ad-hoc positioning system (heavy FP computation)	Arithmetic/Logic unit
Log	Reads log from Flash	Secondary Flash & UART
Multi	Parallel APS computations on both Mote and Stargate	Arithmetic/Logic unit

TABLE II

BENCHMARKS FOR EVALUATING A MOTE AND A STARGATE ENSEMBLE. EACH BENCHMARK STRESSES A PARTICULAR HARDWARE UNIT (SHOWN IN THIRD COLUMN), FOR EXAMPLE, PING UTILITY TESTS THE NETWORK INTERFACE.

Benchmark	μ_{meas}	$\mu_{simulated}$	μ_{meas} - $\mu_{simulated}$	% error $\pm$ 95% conf. bound
adpcmdecode	3.367E+07	3.069E+07	2.980E+06	8.9% $\pm$ 0.28%
adpcmencode	3.068E+07	2.766E+07	3.014E+06	9.8% $\pm$ 0.36%
g721decode	6.272E+08	5.735E+08	5.368E+07	8.6% $\pm$ 0.17%
g721encode	6.527E+08	6.006E+08	5.213E+07	7.9% $\pm$ 0.44%
gsmdecode	1.526E+08	1.420E+08	1.061E+07	7.0% $\pm$ 0.57%
gsmencode	4.335E+08	3.995E+08	3.401E+07	7.8% $\pm$ 0.09%
jpegdecode	2.554E+07	2.235E+07	3.191E+06	12.5% $\pm$ 1.16%
jpegencode	5.412E+07	4.731E+07	6.813E+06	12.5% $\pm$ 0.41%

TABLE III

SIMGATE ACCURACY RESULTS. THE TABLE PRESENTS AVERAGE CYCLE COUNTS FOR MEASUREMENTS AND SIMULATIONS OF MEDIABENCH BENCHMARKS, THE 95% CONFIDENCE INTERVAL ON THE DIFFERENCE BETWEEN THE MEANS, AND THE FRACTION OF THE AVERAGE MEASUREMENT THAT THE INTERVAL CONSTITUTES.

Benchmark	μ_{meas}	$\mu_{simulated}$	μ_{meas} - $\mu_{simulated}$	% error $\pm$ 95% conf. bound
PingShort	9.414299E+07	9.592680E+07	-1.783813E+06	1.9% $\pm$ 6.6%
SenseShort	2.040608E+08	2.051871E+08	-1.126267E+06	0.6% $\pm$ 1.1%
APSShort	1.997744E+08	1.966910E+08	3.083427E+06	1.5% $\pm$ 0.07%
MultiShort	2.128019E+08	2.080650E+08	4.736897E+06	2.2% $\pm$ 1.5%
LogShort	1.637669E+08	1.695956E+08	-5.828771E+06	3.6% $\pm$ 1.25%

TABLE IV

SCENARIO 1 ACCURACY RESULTS. THE TABLE SHOWS THE AVERAGE CYCLE COUNTS FOR MEASUREMENT AND SIMULATION OF THE BENCHMARKS COUPLING SIMGATE WITH SIMULATED MOTE VIA SERIAL LINK. THE FINAL COLUMN SHOWS THE ERROR PERCENTAGE FOR A 95% CONFIDENCE BOUND.

19 degrees of freedom to model the difference of the averages (the $\pm$ bound in the table).

Note that the error percentage and confidence interval also indicate whether we should reject the null hypothesis of equivalence in a two-sided hypothesis test at 95% confidence. If the "margin for error" (confidence interval) spans 0% (i.e. the margin is greater than the error percentage itself), we fail to reject the null hypothesis of equivalence and hence cannot determine whether the observed difference in averages is due to random variation or not. In this experiment, however, the confidence intervals are all quite narrow indicating the the error percentage we observe for each benchmark is statistically significant at the 95% confidence level.

We observe that the accuracy of SimGate for this set of benchmarks is acceptable as a full-system simulation. While error percentages below 5% have been achieved for individual

system components [16], [21], because we simulate the full device (including all parts of the memory hierarchy and the interrupt structure) and run both an operating system and application on it, we expect to introduce additional error. That the maximum error is no more than 13.5% (with 95% confidence) and most of the errors are below 10%, is surprising and is an indication that the simulation is of high quality.

A. Coupled SimGate and SimMote Simulations

To gauge how well SimGate will work in a simulation of a heterogeneous sensor network, we examine its cycle-count accuracy when it is used in conjunction with one or two SimMotes (as described in Section III). Table IV shows the cycle count results for the benchmarks that exercise the Stargate device and the mote that is connected to it via a serial interface (scenario 1). As noted previously, the Stargate device does not support a radio device capable of communicating directly with motes in a sensor network. Instead, it uses mote directly connected to it via a serial interface as a network interface peripheral. These benchmarks are intended to exercise this interaction in a representative way.

The format of Table IV is the same as that described for Table III in the previous subsection. Again, the sample size used to calculate each average is 20 and we compute a 95% confidence interval on the error percentage using a t distribution with 19 degrees of freedom.

Again, the accuracy of the coupled simulation is reasonable for two communicating independent full-device simulations. Note that while the error percentages appear significantly lower than for the SimGate simulation alone, the confidence intervals are also significantly wider. Thus, based on error percentage alone it may appear that the coupled simulations are more accurate. However, there is more relative variation (as we might expect) in the coupled case. As a result, it is the error range, and not the specific error value, that is significant in this case.

For example, consider the results for the *PingShort* benchmark shown in row 1 of Table IV. From the data, it is not possible to determine that the difference between the measured average and simulated average is statistically significant at the 95% confidence level (since the error range spans 0%). However, there is enough variation in both measurements and simulation to make the difference indistinguishable from random variation across an interval that is $\pm 6.6\%$ centered on the observed average.

The *PingShort* benchmark exhibits the widest variation, as indicated by the error range. For the *SenseShort* benchmark the difference in observed average is, once again, statistically undetectable with 95% confidence, but the error range is smaller. In the remaining three cases, there is a statistically significant difference, but both the error percentages and the confidence bounds on those percentages are remarkably small. From this data, we conclude that cycle-counts taken from SimGate when coupled to SimMote via a serial interface, while introducing additional variation, are still reasonably accurate.

The final set of accuracy results we present is for benchmarks that couple SimGate with a SimMote via its serial interface that is then used to communicate with a second SimMote via the radio interface (scenario 2). As described previously, we do not yet know of a mote radio communication simulation that is accurate enough not to overshadow the accuracy (or lack thereof) of SimGate . Thus, these experiments reflect a configuration in which the antenna of the two motes are in physical contact. It is our experience that this configuration eliminates much of the variation resulting from radio communication.

Table V depicts these results using the same format as the in the previous two tables. Similar to the results for *PingShort* and *SenseShort* in Table IV, the additional variation introduced by the second mote and the radio communication makes the difference between observed and simulated averages indistinguishable from random variation at a 95% confidence level. However, the 95% confidence intervals on the error percentage are, once again, similar in magnitude to the error percentages in Tables III and IV for the cases where the averages are significantly different.

From all three tables, then, we conclude that SimGate achieves a similar level of accuracy both when it is used as a single device simulation, and when it is part of a multi-device simulation in which the devices are communicating. Because the software, including the operating system, run by the physical hardware in each of these three experiments is precisely the same as that executed by the simulated devices, we believe that SimGate can be used as an effective tool for estimating Stargate cycle counts in heterogeneous sensor network configurations.

B. SimGate Execution Performance

Since our ultimate goal is to provide a complete sensor network simulation capability that can be used to complement current deployment-based research strategies, the real-time slowdown of SimGate versus the physical hardware is an important consideration. Table VI compares wall-clock timings of the Stargate device to SimGate (t_{cycle}) and to SimGate with the functional simulation enabled ($t_{nocycle}$). For cases where pipeline-simulation is desired, we can enable the parts of SimGate that are necessary to make cycle count estimates internally. Comparing the performance of the resulting functional simulator to the full SimGate simulation gives the cost of achieving the accuracy levels described previously.

The simulator is 10 to 27 times slower than the real hardware when pipeline detailing is not required. This factor is smallest for `adpcm` and `jpeg` (approximately 10 times) and higher for `gsm` and `g741` (approximately 25 times). Cycle-close simulation (r_{cycle}) increases the cost by $2.93X$ (37 to 78 times slower than real hardware). There is a higher variance in these numbers, e.g., `gsm` vs `jpeg`, than for functional simulation ($r_{nocycle}$). One reason for this is cache simulation. The *time required to simulate a cache miss and a cache hit is the same* – although the simulator adjusts the simulated clock and cycle counts appropriately for each.

Benchmark	μ_{meas}	$\mu_{simulated}$	μ_{meas} - $\mu_{simulated}$	% error $\pm$ 95% conf. bound
PingLong	3.228130E+08	3.116003E+08	1.121275E+07	3.5% $\pm$ 2.9%
SenseLong	2.267467E+08	2.254300E+08	1.316726E+06	0.58% $\pm$ 2.1%
APSLong	2.273877E+08	2.212660E+08	6.121661E+06	2.7% $\pm$ 6.3%
MultiLong	2.362925E+08	2.285356E+08	7.756869E+06	3.3% $\pm$ 3.3%
LogLong	1.891255E+08	1.915953E+08	-2.469811E+06	1.3% $\pm$ 2.4%

TABLE V

SCENARIO 2 ACCURACY RESULTS. THE TABLE SHOWS THE AVERAGE CYCLE COUNTS FOR MEASUREMENT AND SIMULATION OF THE BENCHMARKS COUPLING SIMGATE WITH SIMULATED MOTE VIA SERIAL LINK COMMUNICATING WITH A MOTE VIA THE RADIO. THE FINAL COLUMN SHOWS THE ERROR PERCENTAGE FOR A 95% CONFIDENCE BOUND.

Benchmark	t_{meas}	$t_{nocycle}$	t_{cycle}	$r_{nocycle}$	r_{cycle}
adpcmdecode	7.60E-2	1.05E+00	3.23E+00	13.84	42.50
adpcmencode	8.40E-2	1.21E-02	3.61E+00	14.34	42.97
g721decode	1.57E+00	3.70E+01	1.13E+02	23.51	71.73
g721encode	1.64E+00	4.19E+01	1.19E+02	25.45	72.39
gsmdecode	3.82E-01	1.06E+01	2.86E+01	27.65	74.79
gsmencode	1.09E+00	3.01E+01	8.54E+01	27.67	78.64
jpegdecode	6.31E-02	7.28E-01	2.37E+00	11.54	37.61
jpegencode	1.35E-01	2.02E+00	6.18E+00	14.95	45.85

TABLE VI

SIMGATE EXECUTION PERFORMANCE. THE TABLE PRESENTS THE AVERAGE EXECUTION TIME (IN SECONDS) FOR MEASUREMENT (MEAS) AND SIMULATION (W/CYCLE ACCURACY DISABLED (NOCYCLE) AND ENABLED VERSIONS (CYCLE)) OF THE MEDIABENCH BENCHMARKS. THE FINAL TWO COLUMS SHOW THE SLOWDOWN FOR CYCLE-CLOSE SIMULATION AND FUNCTIONAL SIMULATION, RESPECTIVELY.

On a real device a cache hit is much faster than a cache miss. Thus, application memory access patterns can have a large effect on the relative slow down of simulation. We are encouraged by these results since other full system, cycle-accurate, simulations of advanced computer systems executing an OS and application, e.g., SimOS, report slowdowns of $4000X - 6000X$ [18] although the results are not completely comparable since we use different host machines and simulate different targets.

V. RELATED WORK

There is a large body of research on simulation systems. We describe and contrast techniques that are most similar to our work, in particular, frameworks for ensemble sensor device simulation and tools for full system emulation.

A. Ensemble Sensor Network Simulation

There have been a number of significant efforts to simulate and emulate sensor network devices. Most of this prior work has focused on the sensing devices and in particular Mote devices. These projects include Simulavr [9], ATEMU [8], Mule [30] Avrora [11], TOSSIM [6], SensorSim [7], SENS [10] and our ongoing work [17].

The ATEMU and Avrora Mote simulation platforms are most similar to our system. Both provide full-system multi-simulation of Mote devices. However, the multi-simulation enabled by these systems is *homogeneous* – only simulation of Mote devices are coupled and no other sensor network devices, e.g., intermediate nodes, are supported. Both systems use a lock-step method similar to ours to synchronize simulation threads and enable accurate timing and correct communication. ATEMU synchronizes at each cycle and Avrora loosens the synchronization period to thousands Mote cycles. Both ATEMU and Avrora can simulate Motes in real time. Since Avrora is written in Java, its performance is highly dependent on JVM implementation. In our work, we use the similar synchronization technique as in Avrora. However, we must deal with more complex situation in which coordination between devices happens between very different devices.

There are also systems that employ *heterogeneous*, ensemble simulation. In particular, our design vision is similar to the work described in [31]. This prior work describes a comprehensive framework that supports the simulation, and deployment of heterogeneous sensor network systems. This framework uses TOSSIM [6] to emulate Motes and EmStar [32] to emulate "microservers" (a general term for platforms like Stargate). The authors employ a wrapper library, EmTOS, to glue the two simulation systems together by enabling the execution of Mote application on EmStar. In the framework, all applications must be re-compiled and linked to the specific library to be emulated by the system.

In SimGate, our goal is to enable the study, verification, debugging, and analysis of sensor network applications using a simulation platform that does not require any modification to the binaries of the applications or operating system on which

they run. This enables increased flexibility for researchers and ensures that the simulation execution environment is the same as that on the real devices. This SimGate model also enables us to easily obtain important application characteristics (e.g. accurate cycle estimation and interrupt properties) that is more difficult to collect in an emulative environment. Emulation systems do have a speed advantage however. For example, TOSSIM [6] can emulate a Mote 50 times faster that actual Mote execution using a 1.8GHz Pentium IV machine. EmStar can execute re-compiled, microserver code at native speed. In SimGate, we enable users to toggle functional and cycle-close simulation to reduce the overhead of the latter.

B. Full System Simulation

From the perspective of full system simulation and emulation, there are number of software systems that support a wide range of devices [18], [14], [33], [19], [20], [34], [35], [36]. One such, very popular, system is SimOS [18]. SimOS is a full system simulator containing simulation models for most common hardware components, e.g., processor, memory, disk, network interfaces, etc. SimOS features a range of advanced processor models that trade-off accuracy for simulation speed. The fastest model applies dynamic binary translation [37], [33] for maximal simulation speed. The finest-grain model simulates the advanced pipeline structure to provide accurate cycle-level behavior. SimOS is able to simulate the MIPS R4000 processor on a machine with the same architecture, with a slowdown of about $10X$ for binary translation and $5000X$ for detailed pipeline simulation on a SGI 4-processor (150MHz) machine.

SKYEYE (`www.skyeye.org`) is a similar project that simulates a number of ARM-based processors and development boards. SKYEYE also emulates a number of peripherals, including LCD and the Ethernet interface. SKYEYE is based on the GDB ARM emulator which naturally enables the use of gdb as a debugging interface – in much the same way that we do. Although some of the techniques employed in these projects are complementary and useful to our endeavor, these systems are not intended or used for sensor network research. The focus of our work is on a toolset for full-system emulation combined with cycle-close simulation of heterogeneous sensor network devices.

VI. Conclusion

In an effort to make sensor network research more widely accessible we have developed a simulator for functional and cycle-close simulation of intermediate sensor nodes and motes. Our system, called SimGate, implements the complete Intel Stargate device and executes the Linux operating system and XScale applications transparently, without modification.

We investigate the accuracy and efficiency of SimGate in isolation as well as in concert with sensor device (Mote) simulation. Our results indicate that SimGate is functionally correct and enables cycle estimation (if desired) within 9% on average for the benchmarks that we evaluated. When we co-simulate SimGates with SimMotes (our Mote simulator), our system introduces error of less than 4% in all cases. On average, our system is $20X$ slower than a real device when using functional emulation and $58X$ slower when using cycle-close pipeline simulation. We believe that these results indicate that SimGate can be used as an effective tool for accurately simulating Stargate intermediate nodes in heterogeneous sensor network configurations. As part of future work, we plan to investigate techniques for accurate radio and battery modeling, optimization of simulation speed, the scalability of our multi-simulation system for large-scale sensor networks, and simulation of other devices and components.

References

[1] "Habitat Monitoring on Great Duck Island," http://www.greatduckisland.net/index.php.

[2] "Habitat Monitoring on James Reserve," http://www.jamesreserve.edu.

[3] "Ohio State University,Kansei: Sensor Testbed for At-Scale Experiments," Poster, 2nd International TinyOS Technology Exchange, Berkeley, February 2005.

[4] "Mirage: Microeconomic Resource Allocation for SensorNet Testbeds," `https://mirage.berkeley.intel-research.net/`.

[5] G. Werner-Allen, P. Swieskowski, and M. Welsh, "MoteLab: A Wireless Sensor Network Testbed," in *Conference on Information Processing in Sensor Networks: Platform Tools and Design Methods for Network Embedded Sensors (IPSN/SPOTS)*, April 2005.

[6] P. Levis, N. Lee, M. Welsh, and D. Culler, "TOSSIM: Accurate and Scalable Simulation of Entire TinyOS Applications," *ACM Conference on Embedded Networked Sensor Systems*, Nov. 2003.

[7] S. Park, A. Savvides, , and M. B. Srivastava, "SensorSim: a simulation framework for sensor networks," *ACM International workshop on Modeling, analysis and simulation of wireless and mobile systems*, pp. 104–111, 2000.

[8] J. Polley, D. Blazakis, J. McGee, D. Rusk, and J. S. Baras, "ATEMU: A Fine-grained Sensor Network Simulator," *IEEE Communications Society Conference on Sensor and Ad Hoc Communications and Networks*, 2004.

[9] "Simulavr: A simulator for the Amtel AVR processor family," http://www.nongnu.org/simulavr.

[10] S. Sundresh, W. Kim, and G. Agha, "SENS: A Sensor, Environment and Network Simulator," *The IEEE 37th Annual Simulation Symposium*, 2004.

[11] B. L. Titzer, D. K. Lee, and J. Palsberg, "Avrora: Scalable Sensor Network Simulation with Precise Timing," *The Fourth International Symposium on Information Processing in Sensor Networks*, Apr. 2005.

[12] "Stargate: a platform X project," http://platformx.sourceforge.net/.

[13] C. Pereira, J. Lau, B. Calder, and R. K. Gupta, "Dynamic phase analysis for cycle-close trace generation," *In the Proceedings of the 3rd IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis, CODES+ISSS 2005*, Sept. 2005, jersey City, NJ.

[14] "Intel XScale XDB Simulator 2.0," http://www.intel.com/design/pca/prodbref/250424.htm.

[15] D. Brooks, V. Tiwari, and M. Martonosi, "Wattch: a framework for architectural-level power analysis and optimizations," *International symposium on Computer architecture*, pp. 83–94, 2000.

[16] G. Contreras, M. Martonosi, J. Peng, R. Ju, and G.-Y. Lueh, "XTREM: A Power Simulator for the Intel XScale Core," *ACM Conference on Languages, Compilers, and Tools for Embedded Systems*, June 2004.

[17] Y. Wen, R. Wolski, and G. Moore, "DiSenS: Scalable Distributed Sensor Network Simulation," University of California, Santa Barbara, Tech. Rep. CS2005-30, 2005.

[18] M. Rosenblum, S. A. Herrod, E. Witchel, and A. Gupta, "Complete Computer System Simulation: The SimOS Approach," *IEEE Parallel and Distributed Technology*, vol. winter, pp. 34–43, 1995.

[19] P. Magnusson and B. Werner, "Efficient Memory Simulation in SimICS," *Simulation Symposium*, 1995.

[20] P. S. Magnusson, F. Dahlgren, H. Grahn, M. Karlsson, F. Larsson, F. Lundholm, A. Moestedt, J. Nilsson, P. Stenstrm, and B. Werner, "SimICS/sun4m: A Virtual Workstation," *USENIX Technical Conference*, 1998.

[21] T. Austin, E. Larson, and D. Ernst, "SimpleScalar: An Infrastructure for Computer System Modeling," *IEEE Computer*, 2002.
[22] "Intel XScale Technology," http://www.intel.com/design/intelxscale.
[23] "Mica2 sensor board," http://www.xbow.com/.
[24] "MicaZ sensor board," http://www.xbow.com/.
[25] D. Niculescu and B. Nath, "Ad Hoc Positioning System (APS)," *IEEE Global Communications Conference*, Nov. 2001.
[26] M. R. Guthaus, J. S. Ringenberg, D. Ernst, T. M. Austin, T. Mudge, and R. B. Brown, "MiBench: A free, commercially representative embedded benchmark suite," *Workshop on Workload Characterization*, Dec. 2001.
[27] C.Lee, M. Potkonjak, and W. Mangione-Smith, "Mediabench: A tool for evaluating and synthesizing multimedia and communications systems," in *International Symposium on Microarchitecture*, 1997, pp. 330–335.
[28] Y. Wen, S. Gurun, N. Chohan, R. Wolski, and C. Krintz, "Toward Full-System, Cycle-Accurate Simulation of Sensor Networks," University of California, Santa Barbara, Tech. Rep. CS2005-12, 2005.
[29] G. W. Hill, "ACM Alg. 395: Student's T-Distribution," *Communications of the ACM*, vol. 13, no. 10, pp. 617–619, Oct. 1970.
[30] D. Watson and M. Nesterenko, "Mule: Hybrid Simulator for Testing and Debugging Wireless Sensor Networks," in *Workshop on Sensor and Actor Network Protocols and Applications*, Aug. 2004.
[31] L. Girod, T. Stathopoulos, N. Ramanathan, J. Elson, D. Estrin, E. Osterweil, and T. Schoellhammer, "A System for Simulation, Emulation, and Deployment of Heterogeneous Sensor Networks," *ACM Conference on Embedded Networked Sensor Systems*, Nov. 2004.
[32] L. Girod, J. Elson, A. Cerpa, T. Stathopoulos, N. Ramanathan, and D. Estrin, "EmStar: a Software Environment for Developing and Deploying Wireless Sensor Networks," *USENIX Technical Conference*, 2004.
[33] E. Witchel and M. Rosenblum, "Embra: Fast and Flexible Machine Simulation," *ACM SIGMETRICS Performance Evaluation Review*, vol. 24, no. 1, pp. 68–79, May 1996.
[34] R. C. Bedichek, "Efficient Memory Simulation in SimICS," *ACM SIGMETRICS*, 1995.
[35] "QEMU: A Generic and Open Source Processor Emulator," http://fabrice.bellard.free.fr/qemu/.
[36] "The Bochs IA-32 Emulator Project," http://bochs.sourceforge.net.
[37] R. F. Cmelik and D. Keppel, "Shade: A Fast Instruction Set Simulator for Execution Profiling," *ACM SIGMETRICS*, 1994.