# Real-Time System Support for Hybrid Structural Simulation*

David Ferry†, Gregory Bunting‡, Amin Maghareh‡,
Arun Prakash‡, Shirley Dyke‡, Kunal Agrawal†, Chris Gill†, Chenyang Lu†
†Dept. of Computer Science, Washington University, St. Louis, Missouri
dferry@go.wustl.edu, {kunal, cdgill, lu}@seas.wustl.edu
‡Dept. of Civil Engineering, Purdue University, West Lafayette, Indiana
{gbunting, amaghare, aprakas, sdyke}@purdue.edu

## ABSTRACT

Real-time hybrid simulation (RTHS) is an important tool in the design and testing of civil and mechanical structures when engineers and scientists wish to understand the performance of an isolated component within the context of a larger structure. Performing full-scale physical experimentation with a large structure can be prohibitively expensive. Instead, a hybrid testing framework connects part of a physical structure within a closed loop (through sensors and actuators) to a numerical simulation of the rest of the structure. If we wish to understand the dynamic response of the combined structure, this testing must be done in real-time, which significantly restricts both the size of the simulation and the rate at which it can be conducted.

Adding parallelism to the numerical simulation can enable both larger and higher frequency real-time simulations, potentially increasing both the accuracy and the control stability of the test. We present a proof-of-concept exploration of the execution of real-time hybrid simulations (an exemplar of a more general class of cyber-mechanical systems) with parallel computations. We execute large numerical simulations within tight timing constraints and provide a reasonable assurance of timeliness and usability. We detail the operation of our system, its design features, and show how parallel execution could enable qualitatively better experimentation within the discipline of structural engineering.

## Categories and Subject Descriptors

C.3 [**Special-Purpose and Application-Based Systems**]: Real-time and embedded systems; D.1 [**Programming Techniques**]: Parallel programming

*This work supported by NSF grants CNS-1136073 and CNS-1136075


*ESWEEK'14* October 12 - 17 2014, New Delhi, India
ACM 978-1-4503-3052-7/14/10 ...$15.00.
http://dx.doi.org/10.1145/2656045.2656067.

## 1. INTRODUCTION

Civil and mechanical engineers are engaged in developing a new generation of ***smart structures*** — structures capable of withstanding multiple catastrophic events such as earthquakes and tsunamis as well as adapting to normal wear and tear. These structures must be tested and validated under many operating scenarios, but full-scale physical testing of large structures subjected to such extreme loading is often prohibitively expensive. In most cases, only one or a few tests can be performed with each test specimen, since the specimen must be physically reconfigured for new tests or it may be damaged in the course of an experiment.

A solution is to test these designs with ***hybrid testing***, where we augment more manageable physical specimens (e.g. a section of a structure under some experimental condition) with numerical computations simulating the remaining structural aspects. The physical specimen and the simulated portion interact with each other through controllers, sensors and actuators in a closed loop — the simulated structure generates displacement, velocity, and acceleration values that are used to excite the physical specimen through controlled actuation, and the response of the physical specimen, measured by sensors, is then used as an input back into the simulated portion of the structure. In order to test the dynamic behavior of the entire structure faithfully, simulation and control must be integrated in *real-time* to capture the dynamics of the physical specimen (e.g. vibrations that would otherwise damp quickly). In the recent Vision 2020 [17] report developed by the earthquake engineering community, new hybrid simulation capabilities are a major emphasis of the next generation of earthquake engineering research.

Such systems are in many respects representative of an emerging class of embedded computing system, where traditional symmetric multiprocessing hardware architectures are used in the context of computationally demanding cyber-physical applications. This approach is already being used in domains such as autonomous vehicles [27], and it is likely applicable to other areas, as embedded systems must increasingly cope with large amounts of data processing necessary to interact with the world.

Many factors affect the quality of a ***real-time hybrid simulation (RTHS)***. Higher **frequency** tests (the rate at which the simulation is performed) generally provide more fidelity and stability since they are more responsive to changes. More **degrees of freedom** (the distinct number of ways in which the simulated structure is modeled) in the simulation

generally mean that fewer assumptions need to be made about the structure and thus the simulation is more accurate. For larger and/or more complex structures, it is often necessary to use large simulations in order to perform accurate tests. However, there is a tension between performing high-frequency and high degree of freedom tests. Large tests generally require a large amount of computation, while on the other hand, higher-frequency implies that each simulation step must be completed within tight deadlines. Different scenarios require different frequencies; however, for the common case of relatively large, slow moving structures, civil engineers within the RTHS community commonly agree that sensing and actuation at 1024Hz is sufficient for valid real-time tests.

State-of-the-art hybrid real-time testing within earthquake engineering is currently done using sequential tasks on often proprietary platforms [1]. This fundamentally limits the frequency and the complexity of these real-time hybrid simulation experiments, since advances in sequential computational power have essentially plateaued in recent years and most vendors have moved towards producing multicore processors. Large, high-fidelity RTHS computations are unachievable on sequential platforms, so there is a marked need for new systems that support parallel computation with real-time semantics. In order to perform more complex and high-fidelity experiments, we must enable both ***inter-task parallelism*** — multiple tasks running on different processor cores simultaneously, and ***intra-task parallelism*** — individual tasks being able to use multiple processors simultaneously through the use of parallelism within tasks.

In this paper, we investigate the performance of a cyber-physical real-time system tailored for RTHS. This platform is built upon a federated scheduling [31] platform for real-time parallel computations. This federated scheduling platform provides a clean abstraction for programmers to write parallel real-time tasks using existing parallel programming languages (such as Cilk Plus or OpenMP) and enforces the real-time constraints of these tasks. Moreover, our system extends the federated platform to provide additional functionality such as clean and reliable interfaces for intertask communication and hardware communication to support RTHS. By providing a real-time run-time system and a suite of tools to write parallel real-time tasks and manage the intra- and inter-task interactions that arise in the RTHS context our platform allows engineers to iterate more readily on high-level questions of algorithm and experiment design without worrying about real-time mechanisms and programming[1].

Using our RTHS system, we demonstrate the ability to parallelize a variety of RTHS computations, maintain real-time performance and derive benefit from doing so. To support RTHS, our evaluation focuses on real-time numerical simulation which is on the critical path of RTHS experiments and represents the primary challenge in real-time computing for RTHS. In particular, we evaluate the RTHS computing platform through a series of ***virtual RTHS*** (VRTHS) experiments, a form of hardware-in-the-loop testing. In particular, our contributions are as follows:

1. We parallelize a series of virtual RTHS experiments and find that our system for RTHS significantly enhances the capability of RTHS in several key aspects. In particular, we show that ***intra-task parallelism*** can enable both higher-frequency and larger degree-of-freedom simulations. For example, for a realistic model of a nine-story medium-rise building, we can decrease per-period task execution time from 1508$\mu$s to 334$\mu$s on 16 cores. This corresponds to an increase of periodic execution rate from 660Hz to 2990Hz, which allows for RTHS where it was not possible before. We show that for a simple randomized model, we can more than double simulation size.
2. We investigate the possibility of multi-task and multi-rate models (multiple tasks executing at different rates), since the platform also permits ***inter-task*** parallelism. Civil engineers working with RTHS [12] are currently developing techniques to perform this kind of model decomposition, so this is current and timely. We show that our system architecture is able to handle multiple communicating parallel tasks in real-time, and that such decomposition can be particularly useful for executing large models.
3. Our platform for RTHS is fully integrated with a physical shake table and can properly control the shake table's motor and sensors via data acquisition (DAQ) hardware. We characterize the performance of the platform in terms of latencies due to communication with the physical shake-table.

The rest of the paper is as follows. We survey related work in Section 2, and in Section 3 describe our motivating problem: Real-Time Hybrid Simulation. In Section 4 we describe the architecture of our system. In Section 5 we give system performance data and demonstrate the benefit of parallelism for RTHS. Finally, in Section 6 we give directions for future work and conclude.

## 2. RELATED WORK

We now review prior work on computational platforms for hybrid simulation and parallel real-time concurrency.

A variety of packages are used for hybrid simulation in structural engineering. Two notable examples are OpenSees [2] and OpenFresco [3]. The former is a collection of commonly used computational routines (time integration methods, structural models, and so on). OpenFresco is a framework for connecting multiple computational and physical resources. These must be used appropriately within an RTOS to achieve real-time performance (ETS [4] has been used for this purpose in this application domain). OpenSees does support parallelism for a limited class of problems. However, to our knowledge, these technologies have not been used for real-time parallelism. Our work builds upon a package for sequential RTHS called RT-Frame2D [14], which provides structural models and solver methods tailored for RTHS.

Current state-of-the-art RTHS experiments use Mathworks' xPC Target [1] for real-time performance. xPC Target provides MATLAB and Simulink integration, for structural and control engineers to design and prototype cyber-physical computations alongside data acquisition and actuation control. Real-time performance is achieved via rate-monotonic scheduling using hardware interrupt timers. However, this platform is entirely closed source. Additionally, the provided concurrency constructs make it difficult to write efficient parallel computations. Another company, dSPACE, provides TargetLink [5], a similar toolset for deploying Simulink models onto computational platforms. Both these tools were originally designed for programming embedded processors.

[1]Code is available as open source at prt.wustl.edu

In the particular context of RTHS for structural engineering, researchers have realized that it is important to design platforms that enable configurable and scalable experiments [38], but most of the work so far has been done using traditional civil engineering approaches (see [23] for an example). There also has been some work on distributing computational responsibility across multiple platforms in order to get better performance. This can involve coupling a general purpose computing platform alongside specialized hardware [13, 21, 22] (such as a digital signal processor or the xPC Host/Target arrangement), or coupling multiple general purpose platforms [15] over a real-time network or specialized multi-computer shared memory platforms such as SCRAMnet [11]. Most closely related to our work is an earlier middleware design for real-time hybrid testing on Linux [24]. That platform did not support parallelism, however.

Recently, there has been significant interest in parallel real-time scheduling. Many theoretical scheduling algorithms have been designed and analyzed for various task models in both hard real time [9, 10, 16, 26–30, 35, 37] and soft real time [32, 36] contexts. There has not been as much work on building platforms that support real-time parallelism. We are aware of two prior platforms [18, 27]. One of these [27] was used for simulation of a cyber-physical system where they used a special reservation-based OS to implement a system that can run parallel real-time programs for an autonomous vehicle application. In this paper, we use the federated scheduling platform described in [31]. This platform has the advantage that it supports OpenMP and Cilk Plus programs with minimal modifications. In addition, it minimizes interference among different parallel tasks running on the platform. In this work we use this platform for adding parallelism to RTHS experiments by augmenting it in order to allow communication between tasks and between the tasks and the physical specimen through sensors and actuators.

While our primary purpose is not to develop new simulation and verification methodologies, a large body of relevant work exists within the hardware-in-the-loop (HIL) simulation and co-simulation communities. HIL simulation [25] is a technique where embedded control systems and software are developed by replacing a physical prototype with a model of physical dynamics- this allows rapid software development without needing a physical prototype. In our work, VRTHS can be considered a form of HIL simulation. Co-simulation is the practice of integrating multiple software simulators within a single framework, so that large and complex systems (such as a vehicle with mechanical dynamics, thermodynamics, control software, etc.) may be simulated correctly. Some elements of this work, such as multi-rate simulations, do appear in tools for co-simulation such as Flexible Mockup Interface [6]. To our knowledge, such multi-rate simulations have not been performed in real-time with parallelism.

# 3. REAL-TIME HYBRID SIMULATION

Hybrid simulation is a method for coupling numerical simulation with physical experiments in a way that enables low-cost and efficient testing of specimens that would otherwise be impossible or cost-prohibitive to validate. RTHS extends the capability of traditional hybrid simulation by requiring the numerical simulation to be executed with real-time constraints. In this paper, we are considering RTHS applied to the specific problem of structural validation for structural engineering. In this field, researchers develop novel

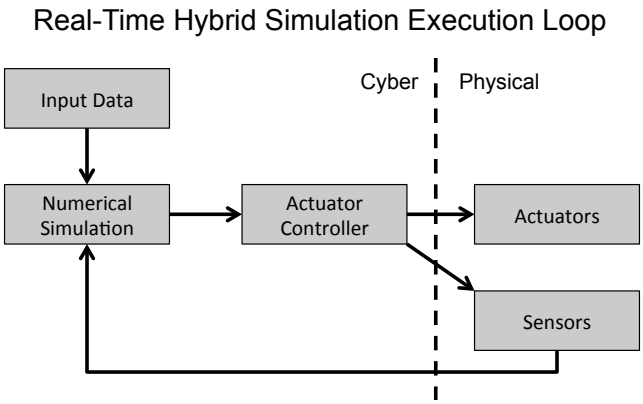


**Figure 1: A standard RTHS control loop for earthquake engineering. Pre-recorded earthquake data is given as the input data to the system. At runtime the structural simulation takes that and sensor data to simulate the structure at large, the results of which are fed into the physical, concurrently running experiment.**

design elements for use in structures such as bridges or office buildings, which may aim to achieve a variety of goals, such as reducing cost or providing earthquake resilience. In any case, such design elements must be validated prior to use in practice.

## 3.1 Hybrid Simulation

Traditionally, engineered structures are validated in two ways: empirically and analytically. For the former, engineers construct a complete replica (potentially a scale model) and subject that replica to physical experimentation in order to verify the structure's response to different loads. For the latter, an analytical model approximates the structure and this model is used to estimate the structure's performance (typically through numerical simulation). Both of these methods have key drawbacks. Physical experimentation is generally expensive, and often infeasible for very large buildings, which require the construction of many mundane elements. On the other hand, a purely analytical validation using numerical simulation is often not possible if reasonable analytical models do not (or could not) exist for the entire structure's behavior, such as when testing damaged structural elements or elements that perform active control [19].

***Hybrid simulation*** combines the strengths and avoids problems of purely physical and purely analytical tests. This technique allows engineers to build physically a small section of a structure to be validated, and then numerically simulate sections whose behavior is predictable. During a test, the simulation is subjected to some hypothetical loading, and this determines what forces would be applied to the specific test specimens under consideration. Actuators then apply those forces physically, and sensors report the results back to the numerical simulation, which are then incorporated in the next simulation step. Thus, the total structural dynamics are captured through a combination of physical testing and numerical simulation. This process is illustrated in Figure 1.

Hybrid simulation also enables interesting cyber-physical polymorphism at the cyber-physical interface. A single physical specimen can represent a variety of tests depending on how it is integrated with different numerical simulations. This reduces the amount of effort spent configuring physical specimens, which is expensive and time consuming.

## 3.2 Extension to RTHS and Virtual RTHS

In traditional hybrid simulation [33, 34], individual simulation steps execute slower than real-time, which fails to capture properties that only appear under dynamic loading (e.g. vibration). For situations where dynamic properties are crucial a traditional simulation cannot help. Instead, engineers and scientists must perform ***real-time hybrid simulation*** where the computation, actuation, and sensing loop must happen in real-time.

One consequence of real-time execution is that faster and more predictable simulation and control algorithms must be used. This potentially reduces accuracy and introduces instability into the control system (i.e. transits the system from unconditional stability to conditional stability). An RTHS control system becoming unstable can result in violent actuation and unintended damage to equipment and physical specimens. To mitigate this risk, RTHS experimentation is always preceded by ***Virtual RTHS***, which replaces the physical specimen with simulated dynamics. Hence, VRTHS is a form of hardware-in-the-loop simulation to verify the correctness of the RTHS execution platform. However, note that VRTHS cannot constitute a full verification- the whole point of RTHS is that the precise dynamics of the physical specimen cannot be modeled effectively, so any VRTHS cannot completely cover all behaviors that might be seen during a real RTHS. However, VRTHS is an effective tool for evaluating the numerical simulation and the software platform that eventually will be used in an RTHS experiment.

Generally, RTHS and VRTHS simulations use fixed step sizes. This is a fundamental feature of the way that RTHS models are cyber-physically coupled to the real world, and allows the RTHS designer to maintain a regular schedule with constant intervals between successive sampling and actuation steps. This is a recognized trade off between simulation accuracy and regularity, and one reason for conducting virtual RTHS prior to full RTHS is that it allows numerical error to be estimated. If the numerical model is predicted to be inaccurate at the desired physical sensing and actuation timescale, then the solution is to run the entire system at a faster rate (such as 2048Hz instead of 1024Hz). For this same reason, RTHS computations favor simple discretizations with regular structure.

# 4. SYSTEM ARCHITECTURE

We have designed a system for constructing and executing RTHS experiments, which allows us to conduct both RTHS and virtual RTHS experiments with inter- and intra-task parallelism. This computational platform is fully integrated with a physical shake-table (Figure 2) and can interact with the physical specimen through sensors and actuators. In this section, we describe both the high-level details of the computational platform, and the methodology used to integrate it with the physical shake-table.

## 4.1 RTHS Platform Overview

The platform for RTHS is built atop Linux with the RT-PREEMPT patch. This allows for numerical simulation and control algorithms to be written using C/C++, gives access to the wide range of Linux services, and allows parallel programs to use general parallel environments such as OpenMP [8] or Cilk Plus [7]. We use the federated scheduling platform [31] to enable parallel real-time behavior. This scheduling algorithm partitions tasks onto processors prior

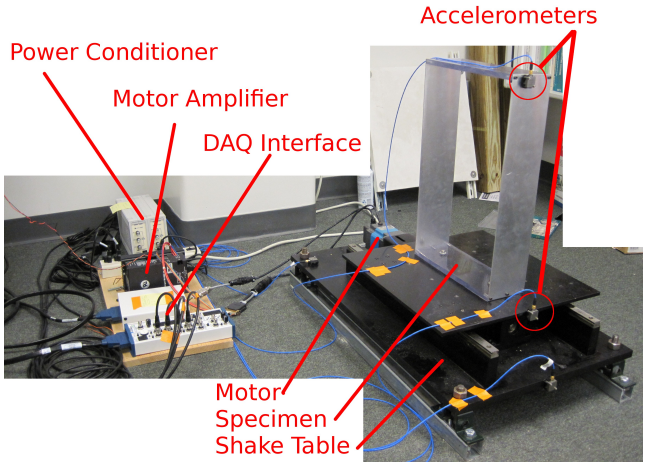


**Figure 2: Our shake table setup. A platform is driven back and forth by a motor and gear assembly. A structure affixed to the platform is instrumented with sensors. During testing, the table is driven by a structural simulation subjected to prerecorded earthquake information, and sensors capture the structure's response.**

to runtime depending on each tasks' processor utilization. ***High utilization*** tasks are those with utilization greater than one (which therefore must exploit intra-task parallelism to meet their deadlines). These are given exclusive use of a suitably large set of processors and as a result experience no contention or interference from any other tasks on the system. ***Low utilization*** tasks have utilization less than one, do not require parallelism to meet their deadlines, and are executed sequentially on the remaining non-exclusive processors with rate monotonic scheduling. The exact formulation of the scheduling strategy and the particulars of its theoretical guarantees are described in [31] and are outside the scope of this paper. This paper does not claim any novelty in the scheduling or execution of parallel real-time tasks, rather we extend the work in [31] with a platform that enables actual RTHS computations by enabling communication between tasks and with the hardware in an efficient and clean interface.

A system overview of our RTHS platform is given in Figure 3. The computational portion of the test consists of several tasks which are either numerical simulation models or control tasks. The figure illustrates two important modifications that adapt a general purpose real-time concurrency platform for conducting RTHS experiments: (1) enabling thread-safe hardware access, and (2) inter-process communication between multiple parallel real-time tasks.

## 4.2 Thread Safe Hardware I/O

One challenge in executing parallel computations for cyber-physical systems is interfacing with hardware that was not designed for use with parallel or even multi-threaded programs. Our data acquisition device drivers explicitly state that they are not thread safe, and in our independent evaluation we found this to be true. Attempting to access a hardware device from multiple threads (even on the same processor) can result in a variety of problems, such as data corruption, intermingling of return values, and segmentation faults.

As Figure 3 illustrates, in order to solve this problem, we used a simple mechanism where all hardware requests are routed though a single thread that is pinned to a reserved

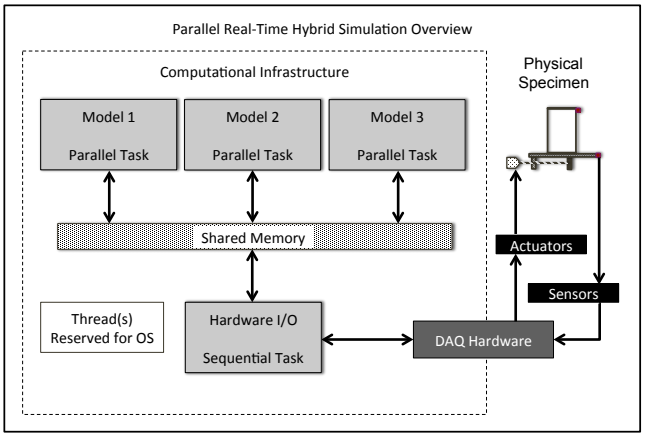


**Figure 3: An overview of our RTHS platform with three numerical models and a dedicated hardware I/O task. Using federated scheduling, we cluster contiguous groups of processors and devote each group to a specific model. The hardware I/O processor interprets simulation results and drives physical actuators via a data acquisition card, which generates and receives hardware signals. Communication between models and the I/O processor is accomplished via writing to shared memory. Note that one processor is reserved for the host operating system.**

processor. All tasks request data from this thread, and this thread polls the data acquisition cards to get the requested data. This design also has the advantage that it reduces the overheads of reads and writes in the parallel tasks themselves and moves it to a separate process. This simple solution is enough for our baseline hardware setup with a small number of sensors and tasks. As we increase the number of sensors and tasks, however, we will investigate more scalable thread-safe strategies for data acquisition as future work.

## 4.3 Interaction Between Tasks

The scheduling theory described in [31] assumes that all tasks are independent. However, tasks in an integrated RTHS experiment must communicate simulation results and sensor data. We added simple functionality for communication between tasks (separate Linux processes) via shared memory.

In general, this communication will happen once per period. Each period all tasks read data from shared memory, execute their periodic iteration, and then wait for all other tasks to finish execution. At the end of each period all tasks write their new data into the pool, wait for all other tasks to finish writing, and then begin executing their next periodic iteration, which starts with reading the previously written data. This easily generalizes to the case of harmonic tasks, but we have not investigated non-harmonic tasks.

For communication between different computational models, we investigated various forms of synchronization: semaphores, barriers, and queues. We find that all these methods have comparable performance (experiments shown in Section 5) and we use the barrier method for our current platform.

# 5. EVALUATION

In this section, we describe the experimental evaluation of our runtime platform for RTHS. In particular, we are interested in how parallelism may enable more degrees of freedom in a simulation, higher frequency experiments, and multi-rate models. We evaluate two virtual RTHS (one realistic, and one contrived). We also wish to understand and quantify the system behavior of our platform, particularly with respect to the robustness of real-time operation.

## 5.1 Experimental Setup

All numerical simulations were conducted on a Dell T7600 16-core machine with two Intel E5-2687W processors. The RTOS we used is the x86_64 Linux kernel version 3.0 with RT-PREEMPT patch rt108. For these tests we disabled hyperthreading, and processor sleep and idle states, as well as processor throttling. Tests are performed with up to fifteen processors, with the remaining one reserved for regular operating system tasks. In some experiments, there is an additional core reserved for an actuator control task. All computations were parallelized with GNU OpenMP, GCC version 4.8.1.

## 5.2 Nine-story Virtual RTHS

Our first Virtual RTHS is a realistic model of a nine-story steel frame structure that typifies medium-rise buildings in Los Angeles. The VRTHS numerical simulation is based on a sequential RTHS software package called RT-Frame2D [14], which provides support for a variety of simulation elements and solvers. The VRTHS was executed for 30,000 periods both with and without parallel concurrency enabled, with up to 15 processors for the parallel case.

At each period, recorded earthquake ground motion is applied to the structural model. Computationally, we are solving a system of primarily linear equations modified by a small number of nonlinear elements. The non linearity is accounted for by recalculating the tangent stiffness matrix at each timestep. This is then added to the linear portions of the problem, resulting in a symmetric sparse matrix with a predicable structure. Solving, we get the displacement, velocity, and acceleration for each element of the structure. An explicit time integration method is used, so no iteration is needed (in most cases implicit is better, but for RTHS it is not practical, as the iterations are too expensive). $\Delta t$ is picked to ensure accuracy, stability, and the ability to meet real time constraints.

For each time step $t$, the major computation is integrating over $\Delta t$ time units by computing the following:

$$K_{t+\Delta t} = \alpha M + \beta C + \tilde{K} \quad (1)$$

$$F_{t+\Delta t} = F_t + [\alpha M + \beta C]\ddot{U}_t + [\gamma M + \delta C]\dot{U}_t \quad (2)$$

$$\Delta U = \{K_{t+\Delta t}\}^{-1} + F_{t+\Delta t} \quad (3)$$

$$U_{t+\Delta t} = U_t + \Delta U \quad (4)$$

$$\Delta \dot{U} = \alpha \ddot{U}_t - \beta \dot{U}_t + \gamma \Delta U \quad (5)$$

$$\dot{U}_{t+\Delta t} = \dot{U}_t + \Delta \dot{U} \quad (6)$$

$$\Delta \ddot{U} = \alpha \Delta \dot{U} - \beta \ddot{U}_t \quad (7)$$

$$\ddot{U}_{t+\Delta t} = \ddot{U}_t + \Delta \ddot{U}, \quad (8)$$

where $U$, $\dot{U}$, and $\ddot{U}$ are displacement, velocity, and acceleration, respectively, $K$ is the global stiffness matrix and $\tilde{K}$ is the nonlinear contribution to the structure stiffness, $F$ is the force exerted on each element, and $M$ and $C$ represent the mass and damping of each element. The Greek lowercase characters represent scalar constants that we have simplified for clarity, and these values change between equations (e.g. the values of $\alpha$ in eq. 1 and eq. 2 are different). For this particular model there are 198 degrees of freedom, so the sizes of the matrices above are 198x198. Detailed information

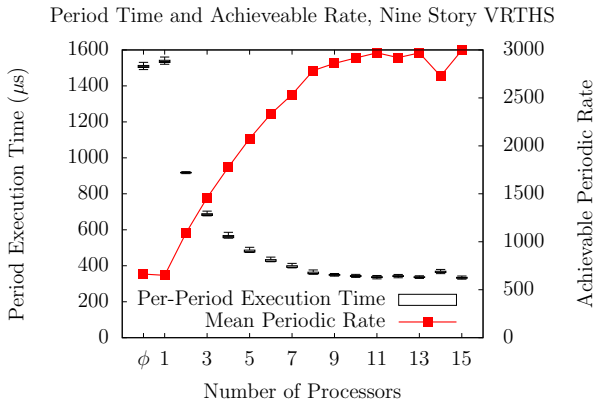


**Figure 4: This graph shows the effect of parallelism on per-period runtimes (boxplot, left axis) and the corresponding achievable periodic rate, derived from the runtime mean (right axis). Due to the large number of samples (30,000 per configuration) this graph has omitted outliers. Minimums and maximums are given in Table 1. $\phi$ denotes serial execution without any parallel runtime.**

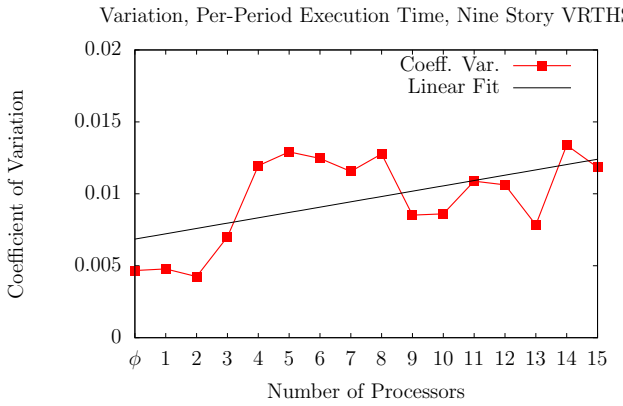


**Figure 5: The coefficient of variation for each processor configuration, calculated as the standard deviation over the mean. $\phi$ denotes serial execution without any parallel runtime.**

about the structural model and executable code can be found in [14] (Model 5, executable RT_F2D_1). The VRTHS consists of both a numerical simulation and physical structure (which is also separately simulated). The computation of this numerical simulation comprises well above 99% of the total computation in the VRTHS, the remainder being the simulation of the physical structure and its dynamics, which is done simply by design. Several of the computational steps are highly parallelizable dense matrix algebra operations.

Figure 4 shows the periodic execution time and the achievable periodic rate for this computation as we change the number of cores. On this particular hardware, using traditional sequential processing, this experiment is only able to achieve a rate of 660Hz, which is significantly lower than 1024Hz that is generally considered safe for these simulations. However, adding parallelism has a significant advantage — using 15 cores, the program achieves a rate of 2990 Hz. This is well above the desired RTHS rate of 1024Hz, so the addition of parallelism has enabled fundamentally new behavior for this model- the ability to be executed in a real-time manner. As described previously, the VRTHS is a full validation of the ability to include this model in a real RTHS experiment. As a result, this nine-story structure (which is an exemplar of the class of medium-rise steel frame buildings) can now be tested via RTHS, where before it could not.

Note that this computation is not exploiting the full parallel capability of the platform — we see that most of the advantage can be extracted using about 8 cores. This is due to the size of the computation and its parallelism — this particular numerical simulation has limited parallelism. An even larger structure would be able to use more cores. Therefore, our platform for RTHS has the capacity to enable even larger RTHS experiments that are impossible to do with sequential execution.

One significant concern of using parallelization in a real-time context is to what extent parallel overheads can affect the real-time behavior of a system. A parallel runtime system must manage and allocate work to threads, and this can induce preemptions and thread migrations, and resultant cache costs, that would not occur in a sequential execution. Since we are layering vanilla (non-real-time) OpenMP on top of a parallel real-time scheduling system, one concern is that there may be significant variability in execution times. In order to allay this concern, we measured the variability. Figure 5 shows the coefficient of variation (CV) of periodic execution times for each processor configuration, which is defined to be the standard deviation of each data set divided by the mean. This yields a unitless quantity that can be used to assess the relative variation of unrelated data sets — for two data sets the one with the higher CV can be thought of as having less predictable behavior. As we can see, the CV does tend to increase as additional cores participate in the computation, but the increase is not significant and stays well within acceptable limits.

One limitation of the CV analysis is that it is relative — as we increase the number of cores given to the computation, the mean execution time decreases. Therefore, the CV increases even when the standard deviation remains the same. Table 1 shows the absolute variability in timing. Quite surprisingly, the absolute variation is lower than sequential variation for all parallel configurations. In essence, increasing parallelism causes periodic execution timings to fall more randomly within some absolute range, but the total extent of that absolute range shrinks, due to the decreased total execution time. Therefore, adding parallelism to a platform does not seem to decrease predictability to a significant extent, at least in our experiments.

## 5.3 Randomized Virtual RTHS

In order to explore the parallel performance of our platform more thoroughly, we now describe a second type of virtual RTHS that permits us to generate hypothetical computations of any desired size. One drawback of the realistic nine-story structure described above, for evaluating our computational system, is that it is a single problem with fixed parameters. As we saw above, it is unable to use the entire power of the platform due to limited parallelism. Using these synthetic benchmarks, we can further understand the actual capacity of the parallel platform.

As above, we can apply recorded earthquake ground acceleration to a simulated structure. Each period, the results of this simulation are fed into actuator control and compensation software, which in turn drive a virtual actuator and virtual structure. The virtual response is then sent back to the numerical simulation. Unlike the previous virtual RTHS,

| Processors | Max. ($\mu$s) | Min. ($\mu$s)e | Difference ($\mu$s) |
|---|---|---|---|
| Serial | 1535.6 | 1490.7 | 44.9 |
| 1 | 1575.3 | 1520.5 | 54.9 |
| 2 | 947.1 | 906.8 | 40.3 |
| 3 | 718.9 | 677.3 | 41.6 |
| 4 | 595.1 | 553.4 | 41.7 |
| 5 | 513.1 | 474.1 | 39.0 |
| 6 | 450.8 | 422.4 | 28.4 |
| 7 | 423.9 | 388.4 | 35.5 |
| 8 | 388.9 | 353.1 | 35.8 |
| 9 | 370.7 | 342.3 | 28.4 |
| 10 | 369.8 | 335.5 | 34.3 |
| 11 | 360.7 | 325.2 | 35.5 |
| 12 | 360.5 | 328.4 | 32.1 |
| 13 | 364.2 | 328.7 | 35.5 |
| 14 | 391.8 | 356.3 | 35.5 |
| 15 | 356.5 | 325.4 | 31.1 |

**Table 1: The maximum and minimum recorded periodic execution times out of 30,000 periods, and their calculated difference. Note that the absolute variation in finishing times (given above as Difference) can actually decrease as we increase parallelism.**

here we use a very simple finite-element model that has no physical significance. Indeed, due to the highly regular structure of this model, the particular values assigned have no effect of computational time.

We represent arbitrary, random dynamic systems using ***first order state space form***, represented by the following equations:

$$y(n) = C * x(n) + D * u(n) \quad (9)$$

$$x(n+1) = A * x(n) + B * u(n), \quad (10)$$

where $n$ is the current simulation step, the vector $u(n)$ is the input to the system, vector $x(n)$ is the current state of the system, vector $y(n)$ is the desired physical output, and $A$, $B$, $C$ and $D$ are static matrices that represent how the states evolve dynamically through time. The above system of equations forms a state space representation of a finite element model for structural dynamics. The vector $u(n)$ is the earthquake ground acceleration (or ground displacement and velocity) at each point in time, while $x(n)$ and $y(n)$ can be thought of as the displacements of structural elements, and the matrices represent the first order differential equations that determine how those elements interact with one another. For this contrived example, we set each matrix to be the same size and square, and say that the ***state size*** of the model is the linear dimension of each matrix. Note that in a real experiment, a single simulated degree of freedom may require multiple states in the matrices.

The primary computational task in these tests is the calculation of equations 9 and 10. We implement the solution of these equations as dense matrix-vector multiplication. This is why the specific structure of our generated buildings (i.e. sparsity) has no impact on execution time. The actuator control scheme is pre-calculated prior to runtime, during which it is reduced to several lookups and multiplications. We simulate the physical structure (the virtualized physical structure, since this is virtual RTHS) as a simple spring modified by a random noise value. These last two calculations are constant time calculations regardless of input size.

This model is undoubtedly simplistic, however, we include it for three reasons. First, it is a good approximation of the types of realistic computations often used in the civil engineering RTHS community [20, 22], since it is completely regular and has highly deterministic behavior. Second, our goal is not to discover efficient finite element discretizations for parallel real-time computation, rather our goal here is to parallelize existing RTHS computations that are used in practice. This model allows us to evaluate our framework for parallelism on a real class of computations being used. Third, this is a general approach for modeling dynamic linear systems, and thus these results are easily extrapolated to many fields beyond RTHS.

We use this randomized VRTHS to explore three advantages of parallelism: (1) increasing the size of the achievable RTHS numerical simulation; (2) increasing the frequency at which the simulation can be performed; and (3) allowing multiple numerical models in a single experiment.

## Enhancing Simulation Size

Increasing the size of a simulation means an increase in the number of internal states in the numerical simulation, which increases the degrees of freedom that the analytical model can effectively simulate, leading to higher accuracy. In our setup, this means performing the first order state updates with larger matrices $A$, $B$, $C$ and $D$ in Equations 9 and 10.

First we ask the question: what is the largest simulation that can be executed within a fixed periodic time window? We determined the size of achievable numerical simulation to the nearest 25 states. The achievable number of states is shown in Figure 6. As the figure shows, the number of states we can achieve increases as we increase the number of cores indicating that parallelism does improve performance.

It is important to note that the improvement is not linear. Even in a perfectly efficient system, we can only expect a sublinear increase in performance (in terms of achievable states) as we add processors to the system. This is because the computational complexity of each simulation step is $O(N^2)$ where $N$ is the number of states, so while additional processors do provide additional capacity, larger models produce a much larger amount of work to perform. In addition, we observed a performance break at 8 and 9 cores, respectively. Most likely this is due to a change in machine operation once it is necessary to involve two physical processors, as opposed to staying entirely within a single processor die.

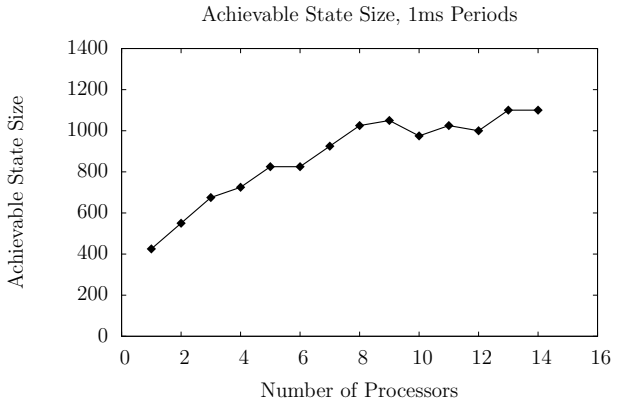


**Figure 6: This plot shows the achievable state space size by number of processor cores allocated to a single simulation task.**

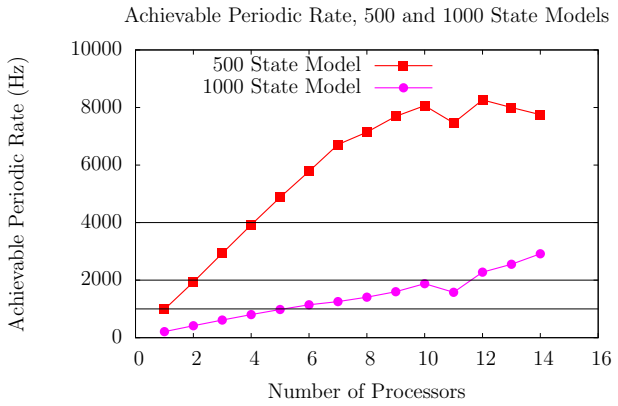


**Figure 7: Achievable periodic rates for two fixed-size models as we increase the number of processing cores available to the simulation task. The horizontal bars show the thresholds for 1KHz, 2KHz, and 4KHz operation.**

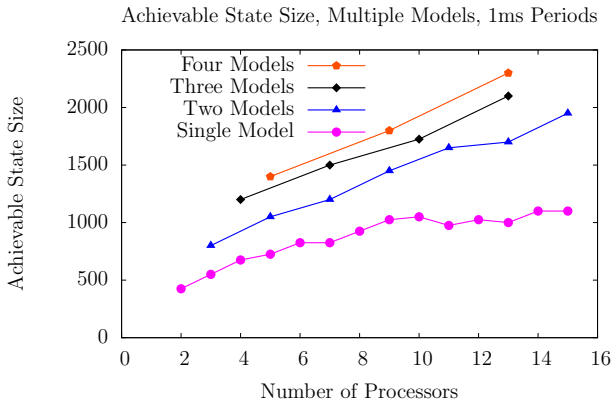


**Figure 8: Multiple model performance. In this arrangement, multiple small models execute simultaneously instead of one large model. Each model rendezvous with a hardware control thread each period, increasing parallel synchronization overhead.**

## Enhancing Simulation Responsiveness

In contrast to increasing simulation size, an alternative measure of performance is to reduce the computation time of numerical simulation, so as to reduce the time step size of the whole system. This provides benefits for simulation stability and also allows engineers to capture effects that might only be detectable at smaller timescales.

We now evaluate a more traditional question: given a fixed amount of work, how fast can it be performed? We take two numerical simulations, containing 500 and 1000 states, and vary the number of processors that participate in parallel computation. In Figure 7 we show how parallelism increases the maximum periodic rate achievable for two fixed size models.

As would be expected, for small models, a diminishing benefit is observed as more processors are used. In the 500 states case, there is no significant performance increase after 7 cores. The 1000 states case is large enough to have continued performance benefit out to 14 cores. We also observe, assuming no overheads, that we are able to achieve 1KHz and 2KHz periodic operation for both simulation sizes, and even 4KHz operation for 500 states. For comparison, we determined that sequential execution is limited to 425 states at 1KHz, and 300 at 2KHz.

## Multi-task Performance

We now investigate the effect of ***inter-task parallelism*** in running multiple simulation models on the same platform. As mentioned earlier, the improvement in the degrees of freedom is not linear due to quadratic increase in work with increasing state size. In general, a greater state size can be achieved if we break larger models into multiple smaller models. For example, running 3 models, each with 500 degrees of freedom is computationally less expensive than running one model with 1500 degrees of freedom. Note that this decomposition is contrived for the purpose of this evaluation. In practice, finding suitable decompositions requires a domain expert capable of doing so. Here, we are using these contrived examples to understand the potential benefits and pitfalls in performance.

Our RTHS platform is well-suited to this context. As mentioned in Section 4, the platform allows multiple simulation tasks seamlessly and will properly partition processors between them according to their utilization. However, now there are multiple simulation tasks executing independently, and each simulation must rendezvous with each other at the start and end of each periodic loop to receive sensor data updates and provide simulation results. Hence, we explore tradeoffs between per-task parallel synchronization overhead and overall system performance.

In this experiment (results shown in Figure 8) for the sake of simplicity, all tasks have the same simulation size. In this chart, the achievable state size is given as the sum of the achievable state sizes in each separate simulation, and the number of cores is the total number of cores used by all simulations plus the core dedicated to hardware I/O and actuator control. For example, the four-tasks data line has three data points: one for 5 cores (four simulations with one core each, plus an I/O core), another for 9 cores (four simulations with two cores each, plus an I/O core) and a third for 13 cores (four simulations with three cores each, plus an I/O core). For the first data point, each of the four simulations achieved 350 states independently, but there are four total simulations, for a total of 1400 states. Figure 8 shows that separating large numerical simulations into smaller simulations does yield an overall performance increase in the size of the state space that can be computed in real-time, and moreover, that the platform can perform their execution and coordinate their communication in real-time.

This type of decomposition in multiple simulation models also enables multi-rate models, in which multiple tasks are simulated at different rates. Not only does this reduce the overall size of a large model, but it can be arranged in such a way that sensitive or important parts of the numerical model are simulated more accurately (given more processing time, or executed at a higher frequency) while other parts are rougher approximations. Figure 9 shows the results of one set of experiments to demonstrate that capability. In these experiments there are three tasks. We assume that Task 1 is a hardware control task with 1 ms period, Task 2 is a task with 1 ms period and Task 3 is a task with 2 ms period. Task 1 always occupies 1 core, and tasks 2 and 3 only communicate a single double-precision floating point value each period. The figure shows the scatter plot of achievable state sizes for Tasks 2 and 3, while the dashed lines show the

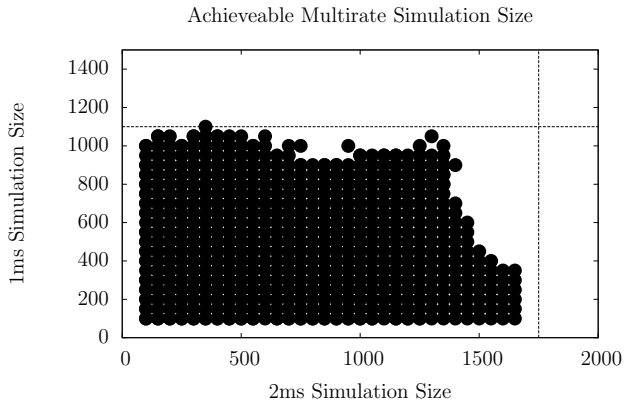


**Figure 9: This plot shows the achievable state sizes for a multi-rate model where a slow model executes alongside a fast model and a third hardware I/O control task. The dashed lines represent the maximum achievable non-multi-rate model sizes.**

maximum achieved state size in the non-multi-rate context (and hence an upper bound on expected performance).

To some degree, increasing the state size of one of these decreases the state size of another since they run on limited resources. However, the trade-off is not linear; in particular, we can run a 1050 state simulation (at 1 ms) at the same time as a 1200 state simulation (at 2 ms). This is primarily because a single simulation experiences diminishing returns after a certain number of cores and adding another simulation is beneficial after this point. There is some price to be paid for multi-socket and multi-rate operation, but it is small and we are able to achieve successful operation across a large range of multi-rate model sizes. This is due in part to how the federated scheduling approach provides a high degree of separation between the multi-rate models.

## 5.4 Communication between Tasks

We now explore the overhead of communication between concurrent tasks in the platform. As mentioned in Section 4, the scheduling strategy and the platform described in [31] assumes that all tasks are independent. However, in RTHS, tasks must communicate with one another. We now characterize these overheads in terms of the size of the communication.

Like multi-rate experiments, we have two tasks: Task 1 (with 1 ms period) and Task 2 (with 2 ms period) communicate with each other every 2 ms. Task 1 is fixed at the state size of 350 and we vary the state size of Task 2. Task 2 always writes data and Task 1 always reads. Table 2 shows the number of states achievable by Task 2. We explored 3 different communication strategies. In the ***semaphore*** method, a semaphore controls access to the shared read/write space. At the start of every period, the writer(s) (large task) grabs the semaphore and writes data. It releases when it's done, and the small task grabs the semaphore to read. In the ***barrier*** method, the writer writes to shared memory while the reader(s) block on a barrier. The writer then releases the reader(s) when done. This does not require the writer to acquire a lock or to block on readers. In the ***queue*** method, the writer writes to a circular buffer and the reader(s) race to keep up. This does not require the writer to block on readers unless they fall so far behind that they saturate the buffers, and readers can fall behind somewhat if they need to. In the ***control*** method, the reader(s) and writer do not synchronize at all. This does not provide correct behavior, but places an upper bound on performance. Table 2 indicates that the type of method used and the total volume of communication had little effect on the performance of the experiment: experiment granularity and these simple methods are adequate at 1ms - 2ms granularity tasks.

| Size (bytes) | Control | Semaphore | Barrier | Queue |
|---|---|---|---|---|
| 8 | 1100 | 1100 | 1100 | 1100 |
| 80 | 1075 | 1075 | 1075 | 1075 |
| 800 | 1075 | 1075 | 1075 | 1075 |
| 8000 | 1075 | 1075 | 1075 | 1075 |
| 80000 | 1075 | 1075 | 1075 | 1075 |
| 800000 | 1000 | 950 | 1000 | 1000 |

**Table 2: Achievable state sizes as influenced by communication sizes and synchronization type. Our specific application uses double width floating point values, which on our machine are 8 bytes, so the first row relates the transfer of one data value, while the last row relates the transfer of 100,000 data values.**

| | Min. ($\mu$s) | Avg. ($\mu$s) | Max. ($\mu$s) |
|---|---|---|---|
| Analog Write | 110 | 113 | 155 |
| Analog Read | 111 | 115 | 170 |
| Digital Read | 53 | 55 | 100 |

**Table 3: Hardware communication overheads as seen by the calling program.**

## 5.5 Overhead of Communicating with a Physical Specimen

Our experimental platform contains two National Instruments data acquisition cards (NI-m6259) capable of sending and receiving analog and digital signals. The motor is driven via an analog output signal through an amplifier. The displacement of the specimen is read through a digital encoder in the form of angular displacement of the motor. Accelerometers provide analog inputs that indicate the current accelerations of the experimental frame. Accessing this hardware through data acquisition devices involves a variety of overheads including calling a proprietary driver with unknown runtime characteristics, and settling times for reading and generating analog voltages. We now quantify these overheads by measuring the time it took to complete various driver calls.

At a minimum, the simplest RTHS experiment will involve one call of each type each period: a digital read to determine the current motor position, an analog write to update the motor speed, and an analog read to measure the impact on the structure. These overheads are given in Table 3. For our target of 1KHz operation, a single threaded application will automatically loose at least 28% of its compute time to hardware I/O under average-case assumptions, or 43% of its compute time under worst-case assumptions. This also motivates our decision to separate all hardware I/O onto a separate thread running on a separate processor. When not busy, this separate thread continually updates sensor information so that there is recent data always available and computational threads do not have to block (or queue) for those results. Conversely, hardware write requests can be merely registered with the I/O thread and the computation thread can then continue on its way. In this way, we reserve a larger portion of each period for computation.

# 6. CONCLUSIONS

We have demonstrated the ability of our platform to provide parallelism that significantly increases the size and speed of numerical simulations for the purpose of real-time hybrid simulation. In a realistic nine-story virtual RTHS experiment, parallelism in our platform enables real time execution when sequential platforms were unable to do so. In addition, we have seen that our real-time system for RTHS enables execution of multiple simultaneous numerical models at multiple rates. While we apply these techniques in the context of structural engineering, these results are broadly applicable to the real-time simulation of other dynamic cyber-physical systems in that RTHS is an exemplar for such. Future extensions of this work are to integrate our platform with a full RTHS experiment, and then use physical measures of experiment performance to determine quantitatively how parallelism improves cyber-mechanical performance.

# 7. REFERENCES


[1] Mathworks xPC, www.mathworks.com/products/xpctarget/.

[2] OpenSees, www.opensees.berkeley.edu.

[3] OpenFresco, www.openfresco.berkeley.edu.

[4] ETS, www.intervalzero.com/products/ets/.

[5] dSPACE TargetLink, www.dspace.com/en/inc/home/products/sw/pcgs/targetli.cfm.

[6] Functional mockup interface for co-simulation. www.fmi-standard.org.

[7] Intel CilkPlus. http://software.intel.com/en-us/articles/intel-cilk-plus.

[8] OpenMP Application Program Interface v3.1. http://www.openmp.org/mp-documents/OpenMP3.1.pdf.

[9] S. Baruah, V. Bonifaciy, A. Marchetti-Spaccamelaz, L. Stougiex, and A. Wiese. A generalized parallel task model for recurrent real-time processes. In *RTSS '12*.

[10] L. Becchetti, M. Dirnberger, A. Karrenbauer, and K. Mehlhorn. Feasibility analysis in the sporadic dag task model. In *ECRTS '13*.

[11] T. Bohman. Shared-memory computing architectures for real-time simulation-simplicity and elegance. Systran Corporation, 1994. Technical Report.

[12] O. Bursi, C. Jia, L. Vulcan, S. Nelid, and D. Wagg. Rosenbrock-based algorithms and subcycling strategies for real-time nonlinear substructure testing. *Earthquake Engineering and Structural Dynamics*, pages 40:1–9, 2011.

[13] J. E. Carrion and B. F. J. Spencer. Model-based strategies for real-time hybrid testing. University of Illinois at Urbana-Champaign, 2007. Technical Report NSEL-006.

[14] N. E. Castaneda Aguilar. *Development and Validation of a Real-Time Computational Framework For Hybrid Simulation of Dynamically-Excited Steel Frame Structures*. PhD thesis, Purdue University, December 2012.

[15] Y. Chae, S. Tong, T. M. Marullo, and J. M. Ricles. Real-time hybrid simulation studies of complex large-scale systems using multi-grid processing. In *Analysis and Computation Specialty Conference*, pages 359–370, 2012.

[16] S. Collette, L. Cucu, and J. Goossens. Integrating job parallelism in real-time scheduling theory. *Inf. Process. Lett.*, 106(5):180–187, 2008.

[17] S. Dyke, B. Stojadinovic, P. Arduino, M. Garlock, N. Luco, J. Ramirez, and S. Yim. 2020 vision for earthquake engineering. Technical report, Report to the National Science Foundation on Recommended Future Directions for Earthquake Engineering Research, 2010. https://nees.org/content/article/206.

[18] D. Ferry, J. Li, M. Mahadevan, K. Agrawal, C. Gill, and C. Lu. A real-time scheduling service for parallel tasks. In *RTAS '13*.

[19] A. Friedman and S. Dyke. Development and experimental validation of a new control strategy considering device dynamics for large-scale MR dampers using real-time hybrid simulation. Technical report, Intelligent Infrastructure Systems Lab Technical Report IISL-003, Purdue University, 2012. https://nees.org/resources/6622.

[20] A. Friedman, S. Dyke, B. Phillips, R. Ahn, B. Dong, Y. Chae, N. Castaneda, Z. Jiang, J. Zhang, Y. Cha, A. Ozdagli, B. Spencer, J. Ricles, R. Christenson, A. Agrawal, and R. Sause. Large-scale real-time hybrid simulation for evaluation of advanced damping system performance. ASCE (submitted April 2013).

[21] X. Gao, N. Castaneda, and S. Dyke. Experimental validation of a generalized procedure for mdof real-time hybrid simulation. ASCE, DOI: 10.1061/(ASCE)EM.1943-7889.0000696.

[22] X. Gao, N. Castaneda, and S. Dyke. Real-time hybrid simulation: from dynamic system, motion control to experimental error. 2012. DOI: 10.1002/eqe.2246.

[23] X. Gao, N. Castaneda, S. J. Dyke, S. Xi, C. Gill, and C. Lu. Experimental Validation of a Scaled Instrumentation for Real-time Hybrid Testing. In *Proceedings of the 2011 American Control Conference*. ACC, June 2011.

[24] H.-M. Huang, T. Tidwell, C. Gill, C. Lu, X. Gao, and S. Dyke. Cyber-physical systems for real-time hybrid structural testing: a case study. In *ICCPS '10*.

[25] R. Isermann, J. Schaffnit, and S. Sinsel. Hardware-in-the-loop simulation for the design and testing of engine-control systems. pages 643–653, 1999.

[26] S. Kato and Y. Ishikawa. Gang EDF scheduling of parallel task systems. In *RTSS '09*.

[27] J. Kim, H. Kim, K. Lakashmanan, and R. Rajkumar. Parallel scheduling for cyber-physical systems: Analysis and case study on a self-driving car. In *ICCPS '13*.

[28] K. Lakshmanan, S. Kato, and R. R. Rajkumar. Scheduling parallel real-time tasks on multi-core processors. In *RTSS '10*.

[29] W. Y. Lee and H. Lee. Optimal scheduling for real-time parallel tasks. *IEICE Trans. Inf. Syst.*, E89-D(6):1962–1966, 2006.

[30] J. Li, K. Agrawal, C.Lu, and C. Gill. Analysis of global edf for parallel tasks. In *ECRTS '13*.

[31] J. Li, J.-J. Chen, K. Agrawal, C. Lu, C. Gill, and A. Saifullah. Analysis of federated and global scheduling for parallel real-time tasks. In *ECRTS'14*.

[32] C. Liu and J. Anderson. Supporting soft real-time parallel applications on multicore processors. In *RTCSA '12*.

[33] S. Mahin, P. Shing, C. Thewalt, and R. Handson. Pseudoynamic test method current status and future directions. *Journal of Structural Engineering*, pages 2113–2128, 1989.

[34] S. A. Mahin and P. B. Shing. Pseudodynamic method for seismic testing. *Journal of Structural Engineering*, pages 1482–1503, 1985.

[35] G. Manimaran, C. S. R. Murthy, and K. Ramamritham. A new approach for scheduling of parallelizable tasks in real-time multiprocessor systems. *Real-Time Syst.*, 15(1):39–60, 1998.

[36] L. Nogueira and L. M. Pinho. Server-based scheduling of parallel real-time tasks. In *International Conference on Embedded Software*, 2012.

[37] A. Saifullah, K. Agrawal, C. Lu, and C. Gill. Multi-core real-time scheduling for generalized parallel task models. In *RTSS '11*.

[38] T. Tidwell, X. Gao, H.-M. Huang, C. Lu, S. Dyke, and C. Gill. Towards Configurable Real-Time Hybrid Structural Testing: A Cyber-Physical Systems Approach. In *Proceedings of the* $12^{th}$ *IEEE International Symposium on Object and Component-Oriented Real-Time Distributed Computing (ISORC): invited paper*. IEEE, Mar. 2009.