ADaPT: An Event-Passing Protocol for Reducing Delivery Costs in Scatter-Gather Parallel Processes

David Woollard†§ Nenad Medvidovic† Walter Yamada§ Theodore Berger§

†Center for Software Engineering
Computer Science Department
University of Southern California
Los Angeles, California 90089-0781

§Laboratory for Neural Dynamics
Center for Biomimetic Microelectronic Devices
University of Southern California
Los Angeles, CA 90089-1451

***Abstract*— The scatter-gather event pattern is commonly used in many high-performance codes to deliver data subsets for use in data-parallel processes and return the results. There are a number of collective algorithms and multiple collective calls in the ubiquitous Message Passing Interface standards which perform aspects of gathering. Programmers have traditionally been left to their own devices, however, to perform complex scatterings of data subsets to processors executing the data-parallel computation. We have developed ADaPT, an Adaptive Data-parallel Publication/Subscription Transport protocol, to reduce the time lost to excessive synchronization with the unicast event-passing commands often employed to scatter data.**

I. Introduction

In recent years, as the performance-to-cost ratio of consumer hardware has continued to increase, computational clusters consisting of fast networks and commodity hardware have become a common sight in research laboratory settings. Unlike much of the software developed for other platforms bearing the moniker *supercomputer*, parallel computations executed on these clusters do not normally rely on one-off parallel languages or compilers. Developers of these systems tend to extend commercially available off-the-shelf approaches to the communications protocols they employ to facilitate parallel code development.

The Message Passing Interface [2], or MPI, is a very common event delivery system used in computational clusters with distributed, or disjoint, memory. In specifying MPI, its contributors intended to facilitate platform independent high-performance parallel codes with a number of point-to-point and collective communications commands. While MPI has gained significant acceptance in the scientific computing and high-performance software communities, multiple inefficiencies have shown themselves, including potentially excessive synchronization of processes in collective communications and lack of support for patterned communications [12].

One prevalent parallel computing paradigm that would benefit from increased support for patterned communications is data-parallel computing. In this paradigm, a large data set is partitioned and distributed, or scattered, between a number of processes performing nearly-identical or identical computations. A single process often initiates the parallel computation and serves as the collection point for the gathering of results. When the optimal aggregation of data results in a number of subsets substantially larger than the number of physical cluster nodes, a sequencing of messages containing the subsets develops and may be exploited.

In order to reduce the delivery costs associated with the repeated delivery of datasets to processes in a multi-part scatter–when the number of datasets to be processed is much larger than the number of processes–we have introduced ADaPT, an Adaptive Data-parallel Publication/Subscription Transport protocol. ADaPT, a protocol based on publish/subscribe, is a two-phase protocol in which processes capable of computing the parallel computation subscribe to the process that serves as the distributor of datasets.

In this paper, we will discuss a computational task faced by the Laboratory for Neural Dynamics which can be made parallel using a data-parallel process, the limitations of the MPI standard to perform a multipart scattering of data, provide an introduction to ADaPT, and discuss further applications of ADaPT to high-performance parallel codes.

II. The Dynamic Synapse Neural Network

The Laboratory for Neural Dynamics is a computational science section of the NSF Engineering Research Center for Biomimetic Microelectronic Systems which combines computational electrophysiology, engineering and other disciplines to develop neural prosthetics. The Laboratory of Neural Dynamics integrates empirically-measured, realistic, and biologically-inspired synapse functional models into artificial neural networks for the purpose of temporal signals processing.

The Dynamic Synapse Neural Network, or DSNN, is a biogically-inspired artificial neural network which the Laboratory uses in natural language and general acoustics processing. First proposed by Berger and Liaw [6], [7], DSNNs offer a remarkable amount of robustness to noisy signal. A key to the functionality of the DSNN model is the balance between a small computational footprint of each biologically-accurate pre- and post-synaptic function and the emerging nonlinearities of network systems composed using these functions.

Similarly to other artificial neural networks implemented in software, each DSNN computation relies on a number of parameters which must be adjusted in order for the network to be properly weighted, or optimized, to recognize input appropriately-similar to the data used in a training process.

The search space consisting of a number of such parameters and their respective variability is far too large for "brute force" methodologies, and so we have explored a number of informed and directed search algorithms.

Currently, the Laboratory optimizes our DSNNs using a genetic algorithm [3]. This search algorithm borrows from its natural analog, producing a *population* of parameter sets, or *genomes*, evaluating the classification abilities of the networks built using these parameter sets, and then *evolving* the fittest parameter sets (i.e. those that produce the best classifications on the training set of inputs) into successive generations in an iterative fashion.

Because of the large variability of signals which the trained DSNN must correctly classify, our training sets are large and classification of the complete set is very costly. Additionally, the interactions of parameters and classification are not completely understood, so many of the genomes *mutated* and *cross-hybridized* from fit members of previous generations are completely unacceptable. Since unacceptability can be determined much more quickly than fitness, each parameter set undergoes testing on a directed subset of the full training set prior to undergoing a complete evaluation of fitness. This process can be described as two-phase evaluation.

III. MPI Implementation

The MPI standard [2] has made great effort to simplify the task of gathering a solution from disparate processes with both the `gather` and `reduce` commands. Because both the aggregation of the dataset to be processed and the mapping of parallel processes are hardware- and computation-dependent, the programmer of high-performance codes is left the decision of how to best parallelize their computation. Though MPI provides a matching `scatter` command, its generic, naive approach to the mapping of datasets to processes addresses neither hardware not computation heterogeneity.

In Figure 1, a case in which MPI's `scatter` command does not yield an optimized time to solution for the training of a DSNN is illustrated. As pointed to by Skjellum, et. al. [11], many implementations of the MPI standards rely on compositions of synchronized point-to-point communications in order to build collective operations such as `scatter` and `gather`. Since the genetic algorithms developed in the Laboratory of Neural Dynamics employ populations of genomes substantially larger than the numbers of physical nodes in the computational clusters with which we train our networks, the need for a multipart scatter arises.

Reliance on the MPI `scatter` yields the process flow seen in Figure 1 where a subset of the genomes are scattered synchronously to each process, evaluated, and gathered again at the main process before the second and third subsets of genomes in the population can be evaluated. As we are using the two-phase evaluation as described in the previous section, a number of processes complete substantially earlier than others. Due to the synchronous nature of the MPI `gather` command, these processes remain idle until all complete, forming an undesirable barrier in the parallel computation at the scattering of each sunset of the population.

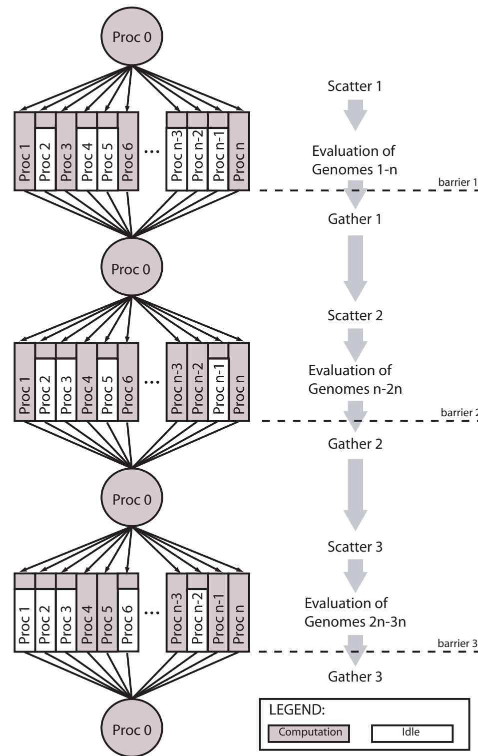

Fig. 1. The two-phase evaluation of a generation of $3n$ genomes by n processes using traditional scatter-gather patterning.

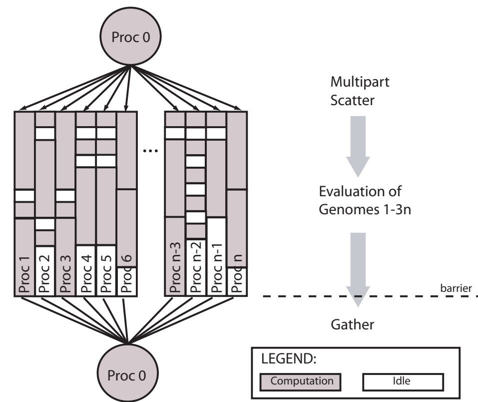

Fig. 2. The two-phase evaluation of a generation of $3n$ genomes by n processes using a multi-part scatter-gather patterning.

A more desirable mapping of evaluations is shown in Figure 2. In this asynchronous scattering, another genome is delivered to each process for evaluation as soon as the process completes its evaluation of the previous genome. This solution is much closer to an optimal time to solution as the time spend idle by each parallel process is minimized. In the next section, we will discuss possible strategies for addressing this asynchronous scattering problem and introduce ADaPT as a possible solution.

IV. Software Architectural Solutions

Within the realm of software architectures [8], [10], there are a number of architectural styles for distributed systems that provide for asynchronous delivery of datasets, including client-server, push-based systems [5], and publish/subscribe [1] (though client-server is traditionally thought of as synchronous, it does provide for separation of channels of communication for each client, asynchronously communicating to each). Both push-based and publish/subscribe are amenable architectures to event-based asynchronous scattering. We will now present ADaPT, an Adaptive Data-parallel Publication/Subscription Transport protocol, which combines features of push-based and publish/subscribe software architectural styles to provide asynchronous scatterings.

A. *ADaPT: A Protocol for Multipart Scatter*

Unlike a uniform scatter, implemented in MPI as main process-centric and initiated as a sequence of synchronous sends with matching receives at each subprocess, ADaPT views each of the parallel processes as independent software components capable of computing on subsets of data. Each subprocess initiates the parallel computation by subscribing to the main process which, in the ADaPT protocol, is a data server. The sequence that emerges from the sending of multiple genomes to each processor can be exploited to reduce the overall processing time at each physical node.

An important distinction between ADaPT and traditional publication/subscription event delivery systems, such as Siena [1], is that unlike traditional pub/sub systems, ADaPT does not duplicate events to service multiple downstream requests. Rather, it distributes events uniquely from a queue in an adaptive round-robin fashion similarly to push-based systems.

Upon receipt of a subscription, the server publishes an event (in our case a genome) to the process. There is another divergence from traditional pub/sub systems at this point. The server waits for another request from the subscribed process before publishing another event to that process. Using data from each subscribed process on its computation time, or μ, the server tracks an average processing time, or $\bar{\mu}$.

Because the protocol is adaptive, when a predetermined number of events have been sent to the subprocesses and a $\bar{\mu}$ has been calculated, the server switches to a second phase of event delivery during which it sends events of the requested type to the process at the regular interval of dictated $\bar{\mu}$ without explicit requests from the server.

Similar to MPI's *eager* protocol, this phase of ADaPT can be too aggressive, potentially flooding the process's buffer. In the event that the number of events in the process's buffer reaches a critical point (such that the buffer might flood upon receipt of another event), the process unsubscribes to the server. After the process has computed each of the events in its buffer, it re-subscribes to the server, starting once again with the first phase of delivery as described above.

Because ADaPT only samples a subset of the computation times at the subprocesses, its calculation of $\bar{\mu}$ is potentially inaccurate. In the next section, we will explore this potential error using a monte carlo simulation of μ values of a large, normalized population of computations.

V. Monte Carlo Simulation

In order to determine a cost in time associated with the ADaPT protocol, it is necessary to determine the probability that a subprocess will unsubscribe from the main process. In order to calculate this probability, the accuracy of the estimated $\bar{\mu}$ must be determined. The probability of unsubscription can then be determined as the probability that a single subprocess capable of buffering m events will receive $m-1$ events of size greater than $\bar{\mu}$.

To determine the error of $\bar{\mu}$, we developed a monte carlo simulation in which random numbers generated from a Park and Miller generator with Bays-Durham shuffling were transformed to a population following the Gaussian distribution using the Box-Muller method [9]. This population is representative of a large population of computation times, or μ.

We repeatedly sampled a percentage of the population. Each population member was sampled uniquely using a linear congruency random number generator for shuffling the population. ADaPT's collection of μ statistics is measured by the mean of this sampling. The plot in Figure 5 illustrates that the error in estimates of the population mean vary inversely with the percentage of the overall population sampled.

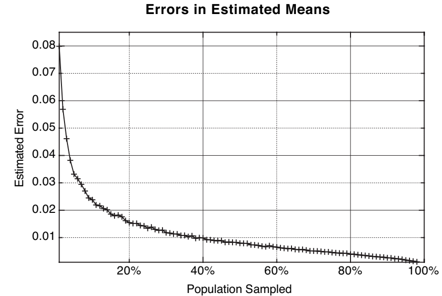

Fig. 3. Standard deviation of error rates in estimates of population mean by random sampling.

From Figure 5, it is clear that the resulting $\bar{\mu}$ error function operates in both a power and linear realm. Since we are interested primary in the sampling of small subsets of the population of parallel computations, we will concern ourselves with the portion of the error function in the power realm. A third-order polynomial fitting of the log-log plot shown in Figure 4 gives us equation 1 for the population of computations, E, with $\mu > \bar{\mu}$.

Defining $s = \log 100(\#sampled/\#computations)$, for $1 \geq s \geq 25$, a polynomial fitting yields the following equation for $\bar{\mu}$:

$$\begin{aligned}\log E &= -0.2693 + (-3.2264e-2)s + \\ & (1.6277e-2)s^2 + (-3.5718e-3)s^3 \\ & \chi^2 = 7.7237e-6 \end{aligned} \tag{1}$$

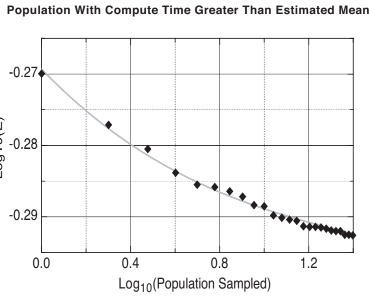

Fig. 4. Third order polynomial fit of the population estimates for the portion of the error function operating in the power realm.

Since we are sampling a gaussian distribution, E, or the percentage of computations in the population with compute time μ greater than the estimated $\bar{\mu}$, is given by the integration of the the area under the normal curve from $\bar{\mu}$ to ∞ (in the worst case this is the actual mean of the population minus the error of the estimated mean). The histogram of this population is illustrated in Figure 5.

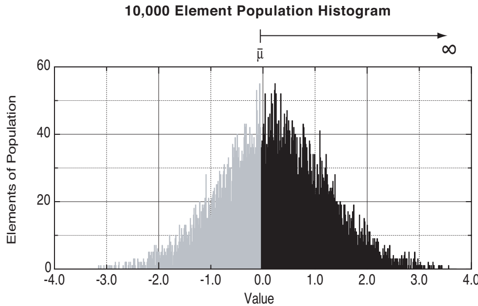

Fig. 5. Gaussian distribution of generated population. $\bar{\mu} \rightarrow \infty$ for 20% random sampling plotted in black.

The probability of unsubscription for a subprocess capable of buffering m events, $P(unsubscribe)$, is given by equation 2.

$$P(unsubscribe) = \frac{\begin{pmatrix} E \times Population \\ m-1 \end{pmatrix}}{\begin{pmatrix} Population \\ m-1 \end{pmatrix}} \quad (2)$$

From this equation, we can see that event-buffering by the computing nodes is an important factor in determining the probability of an unsubscribe. Figure 6 illustrates that, for a 20% sampling rate, the probability of an unsubscribe falls below 1% with an event buffer of size 8 events.

In the next section, we will use the probability of unsubscription given in equation 2 to generate a cost model for event passing in the ADaPT protocol. Additionally, we will present a cost comparison of ADaPT and MPI's `scatter` command when used in a multipart scatter.

VI. Protocol Cost Comparison

We must characterize two aspects of the data-parallel process before we are able to develop a cost model for MPI's `scatter` and ADaPT: event-passing costs as a result of the scatters, and the processing times at each subprocess. We will first analyze the cost of events associated with the scattering of data, or C_s, for MPI and then discuss the cost of the computation, or C_c. Gropp, et. al. present an analysis of MPI's synchronous event transfers in [4]. The cost of MPI's `scatter` in time is given in equation 3.

$$C_s(MPI\ \texttt{scatter}) = (\#Population) \times [3s + r(n + 3e)] \quad (3)$$

Where n is the event payload, e is the envelope (routing information) in bytes, r is the network bandwidth, and s is a measure of network latency. In order to develop a full cost equation for MPI's `scatter`, we must also consider the time of computation. If we consider the best case for MPI's `scatter`, we must sort the parallel processes by process time, or μ. The shortest processes are distributed to the subprocesses first, then the next shortest set, and so on until all data has been processed.

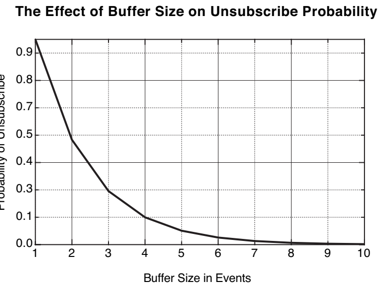

Fig. 6. Message-buffering capability at each processing node effects greatly the probability of an unsubscribe calculated using a 20% sampling rate. See equation 2.

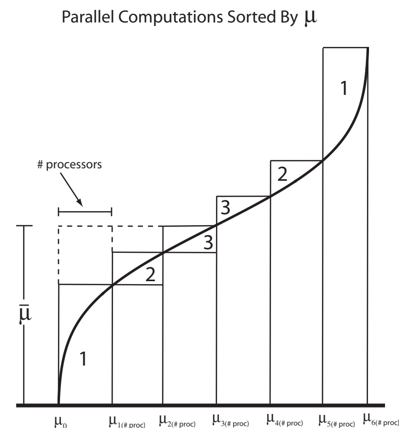

Fig. 7. A sorted plot of μs for each of the parallel processes. Each full bar indicates the time for a synchronized scatter and gather and the portions above the curve indicate idle time.

In figure 7, we have plotted the sorted μs given the normal distribution of process times. The bars represent the process times of each of the scatters. The area of the bar above the

curve represents the idle time for each of the subprocesses (see figure 1). Since the curve plotted in this figure has half-turn symmetry, each of the sections of the plot labeled identically is of equal area. Therefore, the idle, or wasted, time of the synchronous scatter is equal to the average process time multiplied by the number of subprocesses (essentially, the idle time of synchronous scattering will equal an extra scatter in the best case). An equation for $C_c(MPI\ \texttt{scatter})$ is given in equation 4.

$$C_c(MPI\ \texttt{scatter}) = avg(\mu)(\#pop. + \#processes) \quad (4)$$

Using the same conventions as equation 3, the cost of ADaPT's scatter in time is given by the equation 5.

$$\begin{aligned} C_{s:subscription}(ADaPT) &= (\#subprocs.) \times [s + re] \\ C_{s:phase1}(ADaPT) &= (\#samps.) \times \\ &\quad [2s + r(n + 2e)] \\ C_{s:phase2}(ADaPT) &= (\#pop. - \#samps.) \times \\ &\quad [s + r(n + e)] \end{aligned} \quad (5)$$

In the above equation for $C_s(ADaPT)$, the first phase of delivery is more expensive due to the cost associated with receiving feedback on a percentage of the population of data in order to estimate $\bar{\mu}$. In the second, or aggressive, phase of the ADaPT protocol, events are simply sent to each subprocess at the regular interval $\bar{\mu}$. In addition, there is a cost associated with subscribing to the the main process.

The computation time associated with the ADaPT protocol is given as:

$$C_c(ADaPT) = avg(\mu)(\#pop.) \quad (6)$$

One final cost that must be considered for the ADaPT protocol is the cost associated with unsubscribing from the main process, or $C_{unsubscribe}$. The process of unsubscribing is accomplished by sending an unsubscribe message to the main process, processing the events in the buffer, and sending a resubscribe message to the main process. The cost of unsubscription (with probability P as given in equation 2) is given in equation 7.

$$\begin{aligned} &C_{unsubscribe}(ADaPT) = \\ &P(unsubscribe)\left[2(s + re) + (m-1)\bar{\mu}\right] \end{aligned} \quad (7)$$

Composing the costs of event-passing and process time for MPI's `scatter` and idealizing the network to remove the latency term s, we have determined the best-case time to completion, or TTC, for a multiscatter data-parallel process using MPI `scatter` given in equation 8.

$$\begin{aligned} &TTC(MPI\ \texttt{scatter}) = \\ &(\#pop.)[r(n + 3e)] + (\#pop. + \#subprocs.)\bar{\mu} \end{aligned} \quad (8)$$

Time to completion for an ADaPT-based multiscatter data-parallel process is given in equation 9.

$$\begin{aligned} &TTC(ADaPT) = (\#subprocs.)[re] + \\ &(\#samps.)[r(n+2e)] + (\#pop. - \#samps.)[r(n+e)] + \\ &(\#pop.)\bar{\mu} + (\#pop./m)P(unsubscribe)[2re + (m-1)\bar{\mu}] \end{aligned} \quad (9)$$

VII. Analysis

We shall assume, for illustrative purposes, that the number of samplings taken is the same as the number of subprocesses working of the data-parallel computation (i.e., only one compute cycle is monitored for each subprocess). Comparing the extra event-passing and idle subprocessing time of MPI's `scatter` with the costs of unsubscribing in ADaPT, we can determine the more appropriate protocol for use in a given data-parallel computation. This comparison is given in equation 10.

$$\begin{aligned} &(\#pop. - \#samps.)2re + (\#samps.)\bar{\mu} \gtreqless \\ &(\#pop./m)P(unsubscribe)[2re + (m-1)\bar{\mu}] \end{aligned} \quad (10)$$

A number of conclusions can be drawn about the general applicability of the ADaPT protocol from the above equation. First, when m is large, ADaPT will easily out perform MPI's `scatter` due to the greatly-reduced probability of unsubscription (see figure 6). Second, assuming $\bar{\mu} \gg re$ (process time is much greater than the time to transfer an empty event across the network), then the hidden savings in the ADaPT protocol is the reduction of idle time at each of the processing nodes, not the reduced amount of event traffic on the network. These conclusions are supported by the plot of this inequality in figure 8.

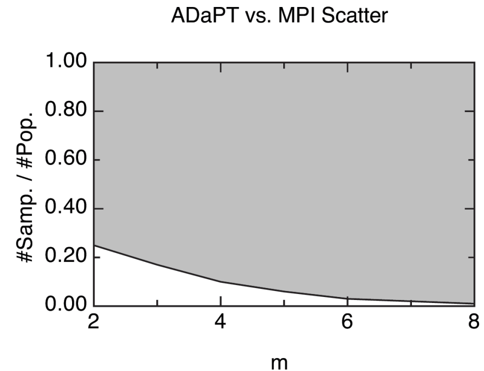

Fig. 8. For a population size of 10,000, the inequality in equation 10 is plotted. The grey area represents ADaPT's outperformance of MPI's `scatter`.

From figure 8, for larger event buffers, $m > 8$, ADaPT always outperforms MPI's `scatter` in the time to completion of a multipart scatter. In the case that a developer cannot buffer eight events at each subprocess, a larger initial sampling is required in order for ADaPT to outperform MPI.

VIII. Conclusions

A multipart scattering is required when the aggregation of large datasets for use in data-parallel processes results in the mapping of multiple data subsets to each subprocess such as in the training of our DSNNs. Inefficiencies in the MPI `scatter` protocol result in excessive synchronization at each scattering of a number of data subsets to the subprocesses.

In order to facilitate asynchronous multiscatters, we have developed ADaPT, a two-phased adaptive protocol which approximates the average runtime of each computation of the subsets of data. Using a monte carlo simulation, we determined the probability that a subprocess using ADaPT will unsubscribe from the main process, causing a delay in the system as the subprocess computes the events in its local buffer before resubscribing.

Additionally, we have analyzed inefficiencies in multiple uses of MPI's `scatter` command to distribute data subsets and we provide a cost comparison between MPI's `scatter` and ADaPT. As the buffer size at each subprocess is increased, the probability of unsubscription is decreased, and the costs associated with ADaPT are minimized.

By applying software architectural principles to event-passing in computational clusters, we have developed a new protocol for event transfers which optimizes multipart scatterings of datasets, a common practice in high-performance and scientific parallel codes.

References

[1] A. Carzaniga, D. S. Rosenblum, and A. L. Wolf. Design and evaluation of a wide-area event notification service. *ACM Transactions on Computer Systems*, 19(3):332–383, 2001.

[2] M. P. I. Forum. MPI: A message-passing interface standard. Technical Report UT-CS-94-230, 1994.

[3] D. E. Goldberg. *Genetic Algorithms in Search Optimization and Machine Learning*. Addison-Wesley, 1989.

[4] W. Gropp, E. Lusk, and A. Skjellum. *Using MPI: Portable Programming with the Message Passing Interface*. MIT Press, 1999.

[5] M. Hauswirth and M. Jazayeri. A component and communication model for push systems.

[6] J.-S. Liaw and T. W. Berger. Dynamic synapse: A new concept of neural representation and computation. *Hippocampus*, 6:591–600, 1996.

[7] J.-S. Liaw and T. W. Berger. Dynamic synapse: Harnessing the computing power of synaptic dynamics. *Neurocomputing*, 26–27:199–206, 1999.

[8] D. E. Perry and A. L. Wolf. Foundations for the study of software architectures. *ACM SIGSOFT Software Engineering Notes*, October, 1992.

[9] W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery. *Numerical Recipes in C*. Cambridge University Press, 1988.

[10] M. Shaw and D. Garlan. *Software Architecture: Perspectives on an Emerging Discipline*. Prentice-Hall, 1996.

[11] A. Skjellum. High performance mpi: Extending the message passing interface for higher performance and higher predictability, 1998.

[12] A. Skjellum, P. Bangalore, J. Gray, and B. Bryant. Reinventing explicit parallel programming for improved engineering of high performance computing software.