Modelling the Runtime of the IQMR Method for Large and Sparse Linear Systems on Parallel Computers

Lawrence Tianruo Yang *
Department of Computer Science, St. Francis Xavier University
P. O. Box 5000, Antigonish, B2G 2W5, Nova Scotia, Canada

Abstract

For the solutions of large and sparse linear systems of equations with unsymmetric coefficient matrices, we propose the IQMR method by using the Lanczos process as a major component combining elements of numerical stability and parallel algorithm design. The algorithm is derived such that all inner products and matrix-vector multiplications of a single iteration step are independent and communication time required for inner product can be overlapped efficiently with computation time of vector updates. Therefore, the cost of global communication on parallel distributed memory computers can be significantly reduced. In this paper, we show the execution time of collective communication operations in PVM and PMI can be modelled by runtime functions in closed form on CrayT3D and CrayT3E. We also demonstrate that the runtime functions can be used to model the computation and communication of the IQMR method. Some theoretical and experimental results will be presented and compared as well.

1 Introduction

One of the fundamental tasks of numerical computing is the ability to solve linear systems with unsymmetric coefficients. These systems arise very frequently in scientific computing, for example from finite difference or finite element approximations to partial differential equations, as intermediate steps in computing the solution of nonlinear problems or as subproblems in linear and nonlinear programming.

One of them, the quasi-minimal residual (QMR) algorithm [12], uses the Lanczos process [11] with look-ahead, a technique developed to prevent the process from breaking down in case of numerical instabilities, and in addition imposes a quasi-minimization principle. This combination leads to a quite efficient algorithm among the most frequently and successfully used iterative methods. This method is widely used for very large and sparse problems, which in turn are often solved on massively parallel computers.

On massively parallel computers, the basic time-consuming computational kernels of QMR are usually: inner products, vector updates, matrix-vector multiplications. In many situations, especially when matrix operations are well-structured, these operations are suitable for implementation on vector and share memory parallel computers [10]. But for parallel distributed memory machines, the matrices and vectors are distributed over the processors, so that even when the matrix operations can be implemented efficiently by parallel operations, we still cannot avoid the global communication, i.e. communication of all processors, required for inner product computations. Vector updates are perfectly parallelizable and, for large sparse matrices, matrix-vector multiplications can be implemented with communication between only nearby processors. The bottleneck is usually due to inner products enforcing global communication. The detailed discussions on the communication problem on distributed memory systems can be found in [6, 8]. These global communication costs become relatively more and more important when the number of the parallel processors is increased and thus they have the potential to affect the scalability of the algorithm in a negative way [6, 8].

Recently, we have proposed a new improved two-term recurrences Lanczos process [19] without look-ahead as the underlying process of QMR. The algorithm is reorganized without changing the numerical stability so that all inner products and matrix-vector multiplications of a single iteration step are independent and communication time required for inner product can be overlapped efficiently with computation time. Therefore, the cost of global communication on parallel distributed memory computers can be significantly reduced. The resulting IQMR algorithm main-
tains the favorable properties of the Lanczos process
while not increasing computational costs.

Now, MPI is probably the most commonly used
message passing library for programming distributed
memory parallel computers. Implementations of MPI
are available for all commercially available parallel ar-
chitectures and are generally accepted by application
programmers. But to find an efficient parallel version
of a scientific and engineering algorithm such as the
IQMR method, still takes a lot of time for program-
ning and measuring different versions of the algorithm.
This is due to the fact that scientific and engineering
algorithms on parallel computers show a complicated
time behavior caused by the overheads of explicit
communication or synchronization and load imbalance.
In this article, we first investigate and compare single-
transfer and collective communication operations both
in isolation and in the context of large message pass-
ing application programs. We will then look at mainly
the collective communication operations such as single-
broadcast which is mainly used for modelling the com-
mutation cost of inner products. For the operation we
consider several variants being realized in different
ways within the PVM or MPI environment. The run-
time prediction provides functions in closed form in the
spirit of the function for single-to-single transfer linear
in the message size, which is valid for a large number of
machines. The functions depend on several parameters
including application as well as machine parameters.
The values of the parameters are determined with the
least-squares method. The resulting runtime formu-
las describe the communication time quite accurately.
The results of the experiments and the modelling can
be used in compiler tools, in parallelizing compilers or
directly by the application programmer. A knowledge
of the runtime behavior of such communication opera-
tions can have a great influence on the design process
of the efficient parallel IQMR method. The integration
of the runtime formulas into a cost model allows us to
predict the runtimes for the entire IQMR method quite
accurately.

The paper is organized as follows. We will describe
briefly the improved Lanczos process and the result-
ing improved quasi-minimal residual (IQMR) method
in section 2, 3 and 4, respectively. In section 6, the
parallel performance model is presented including the
communication model and assumptions for computa-
tion time and communication costs. Particularly we
will describe the modelling of execution time of col-
lective communication operations in PVM and MPI on
CrayT3D and Cray T3E parallel systems. The corre-
sponding theoretical complexity analysis will be pre-
sented as well. Finally we offer some preliminary mea-
sured and estimated timing results and conclusions, re-
spectively.

2 The Original Lanczos Process

The classical unsymmetric Lanczos process [15]
based on three-term recurrence reduces a matrix \( A \) to
a tridiagonal form \( T \) using a similarity transforma-
tion which leads to the following three relations that serve
to derive the unsymmetric Lanczos process:

\[
W^T V = I, \quad AV = VT, \quad A^T W = W^T T,
\]

where \( I \) is the identity matrix and \( T \) is a tridiagonal
matrix. More precisely, this process, starting with two
vectors \( v_1 \) and \( w_1 \) satisfying \( w_1^T v_1 = 1 \), iteratively
generates two finite sequences of vectors \( v_n \) and \( w_n \) such
that, for \( n = 1, 2, \ldots \)

\[
K_k(v_1, A) = \text{span}\{v_1, v_2, \ldots, v_n\},
K_k(w_1, A^T) = \text{span}\{w_1, w_2, \ldots, w_n\},
\]

and the two sets are biorthogonal as follows

\[
w_m^T v_n = \begin{cases} 0 & m \neq n, \\ 1 & m = n \end{cases}
\]

We denote by

\[
V = [v_1, v_2, \ldots, v_n] \quad \text{and} \quad W = [w_1, w_2, \ldots, w_n],
\]

the matrices containing the Lanczos vectors \( v_n \) and \( w_n \)
as columns.

This process leads to two inner products per itera-
tion. The inner product requires global communication
on parallel distributed memory computers. Some im-
provements on the reduction of global communication
required by inner products have been investigated in
[14].

3 The Improved Lanczos Process

Although Lanczos used a similar technique built on
coupled two-term recurrence in the early of 1950’s,
most published papers have been dealing with the
three-term recurrence process, until, recently, Freund
et al. [13] reused this idea to improve numerical stabili-
ty. They claimed that, the latter variant of the Lan-
czos process, may be numerically more stable. That is
why we pursue further on this unsymmetric Lanczos
process with two-term recurrences as underlying pro-
cess of the QMR method.

Recently, Bückler et al. [1, 2] proposed a new parallel
version of the QMR method based on the coupled two-
term recurrences Lanczos process without look-ahead
strategy. The algorithm is derived such that both se-
quencies of generated Lanczos vectors are scalable and
there is only one single global synchronization point per iteration. Based on the similar idea, we present a new improved two-term recurrences Lanczos process without look-ahead technique.

Algorithm 1 Improved Lanczos Process
1: \[ p_0 = \psi_0 = u_0 = 0, \quad \gamma_1 = (\hat{v}_1, \hat{v}_1), \rho_1 = 0, \xi_1 = (\hat{w}_1, \hat{w}_1), \]
2: \[ s_1 = A^T \hat{w}_1, \rho_1 = (\hat{w}_1, \hat{v}_1), \varepsilon_1 = (s_1, \hat{v}_1), \tau_1 = \hat{\varepsilon}_1. \]
3: \text{for } n = 1, 2, \ldots \text{ do}
4: \[ q_n = \frac{1}{\xi_n} s_n - \frac{\rho_n}{\xi_n} \rho_n; \]
5: \[ \hat{w}_{n+1} = q_n \hat{w}_n; \]
6: \[ s_{n+1} = A^T \hat{w}_{n+1}; \]
7: \[ t_n = A \hat{\varepsilon}_n; \]
8: \[ u_n = \frac{1}{\tau_n} u_n - \mu_n u_{n-1}; \]
9: \[ \hat{v}_{n+1} = u_n; \]
10: \[ \rho_n = \frac{1}{\gamma_n} \rho_n - \mu_n \rho_{n-1}; \]
11: \[ \gamma_{n+1} = (\hat{v}_{n+1}, \hat{v}_{n+1}); \]
12: \[ \xi_{n+1} = (\hat{w}_{n+1}, \hat{w}_{n+1}); \]
13: \[ \rho_{n+1} = (\hat{w}_{n+1}, \hat{v}_{n+1}); \]
14: \[ \varepsilon_{n+1} = (s_{n+1}, \hat{v}_{n+1}); \]
15: \[ \mu_{n+1} = \frac{2 \mu_n \varepsilon_{n+1}}{\gamma_{n+1} \rho_{n+1}}; \]
16: \[ \tau_{n+1} = \frac{\xi_{n+1} \rho_{n+1} - \gamma_{n+1} \mu_{n+1}}{\rho_{n+1}}; \]
17: \text{end for}

We reschedule the computation within an iteration step, without affecting the numerical stability, such that all inner products and matrix-vector multiplications of a single iteration step are independent and the communication time required for inner product can be overlapped efficiently with computation time. The framework of this improved Lanczos process based on two-term recurrences is described in Algorithm 1.

The improved Lanczos process can be efficiently parallelezed as follows:

- The inner products of a single iteration step (11), (12), (13) and (14) are independent.
- The matrix-vector multiplications of a single iteration step (6) and (7) are independent.
- The communications required for the inner products (11), (12), (13) and (14) can be overlapped with the update for \( p_n \) in (10).

Therefore, the cost of communication time on parallel distributed memory computers can be significantly reduced.

4 The Improved Quasi-Minimal Residual Method

The improved Lanczos process now is used as a major component to a Krylov subspace method for solving a system of linear equations

\[ Ax = b, \quad \text{where } A \in \mathbb{R}^{n \times n} \text{ and } x, b \in \mathbb{R}^n. \]  

In each step, it produces approximation \( x_n \) to the exact solution of the form

\[ x_n = x_0 + K_n(r_0, A), \quad n = 1, 2, \ldots \]

Here \( x_0 \) is any initial guess for the solution of linear systems, \( r_0 = b - Ax_0 \) is the initial residual, and \( K_n(r_0, A) = \text{span} \{r_0, Ar_0, \ldots, A^{n-1}r_0\} \), is the \( n \)-th Krylov subspace with respect to \( r_0 \) and \( A \).

Algorithm 2 Improved Quasi-Minimal Residual Method
1: \( \hat{v}_1 = \hat{w}_1 = r_0 = b - Ax_0, \lambda_1 = 1, \kappa_0 = 1, \mu_1 = 0, \)
2: \[ p_0 = \psi_0 = u_0 = d_0 = f_0 = 0, \quad \gamma_1 = (\hat{v}_1, \hat{v}_1), \]
3: \[ s_1 = A^T \hat{w}_1, \rho_1 = (\hat{w}_1, \hat{v}_1), \varepsilon_1 = (s_1, \hat{v}_1), \tau_1 = \hat{\varepsilon}_1. \]
4: \text{for } n = 1, 2, \ldots \text{ do}
5: \[ q_n = \frac{1}{\xi_n} s_n - \frac{\rho_n}{\xi_n} \rho_n; \]
6: \[ \hat{w}_{n+1} = q_n \hat{w}_n; \]
7: \[ s_{n+1} = A^T \hat{w}_{n+1}; \]
8: \[ t_n = A \hat{\varepsilon}_n; \]
9: \[ u_n = \frac{1}{\tau_n} u_n - \mu_n u_{n-1}; \]
10: \[ \hat{v}_{n+1} = u_n; \]
11: \[ \rho_n = \frac{1}{\gamma_n} \rho_n - \mu_n \rho_{n-1}; \]
12: \[ \gamma_{n+1} = (\hat{v}_{n+1}, \hat{v}_{n+1}); \]
13: \[ \xi_{n+1} = (\hat{w}_{n+1}, \hat{w}_{n+1}); \]
14: \[ \rho_{n+1} = (\hat{w}_{n+1}, \hat{v}_{n+1}); \]
15: \[ \varepsilon_{n+1} = (s_{n+1}, \hat{v}_{n+1}); \]
16: \[ \mu_{n+1} = \frac{2 \mu_n \varepsilon_{n+1}}{\gamma_{n+1} \rho_{n+1}}; \]
17: \[ \tau_{n+1} = \frac{\xi_{n+1} \rho_{n+1} - \gamma_{n+1} \mu_{n+1}}{\rho_{n+1}}; \]
18: \[ \theta_n = \frac{\tau_n (1 - \lambda_n)}{\lambda_n \tau_n + \gamma_n}; \]
19: \[ \kappa_n = \frac{\gamma_n \tau_n (1 - \lambda_n)}{\lambda_n \tau_n + \gamma_n}; \]
20: \[ \lambda_n = \frac{\lambda_n \tau_n + \gamma_n}{\lambda_n \tau_n + \gamma_n}; \]
21: \[ d_n = \theta_n d_{n-1} + \kappa_n p_n; \]
22: \[ f_n = \theta_n f_{n-1} + \kappa_n u_n; \]
23: \[ x_n = x_n-1 + d_n; \]
24: \[ r_n = r_{n-1} - f_n; \]
25: \text{end if}
26: \text{end for}

Given any initial guess \( x_0 \), the \( n \)-th Improved QMR iterate is of the form

\[ x_n = x_0 + V_n z_n, \]  

where \( V_n \) is generated by the improved Lanczos process, and \( z_n \) is determined by a quasi-minimal residual property.
Based on the similar idea in [1, 2] with computation rearrangement, we derive an improved QMR method (IQMR) based on coupled two-term recurrences with scaling of both sequence of Lanczos vectors for maintaining the numerical stability. All inner products and matrix-vector multiplications of a single iteration step in the IQMR are independent and communication time required for inner product can be overlapped efficiently with computation time. The framework of this improved QMR method using Lanczos algorithm based on two-term recurrences as underlying process is depicted in Algorithm 2.

The improved QMR method using Lanczos algorithm as underlying process can be efficiently parallelized as follows:

- The inner products of a single iteration step (12), (15), (16), (17) and (18) are independent.
- The matrix-vector multiplications of a single iteration step (7) and (8) are independent.
- The communications required for the inner products (12), (15), (16), (17) and (18) can be overlapped with the update for \( p_{n} \) in (11).

Therefore, the overhead of communication on parallel distributed memory computers can be significantly reduced.

5 Parallel System Overview

The parallel computing systems we mainly use are the Cray T3D and T3E distributed memory multiprocessors (DMMs) with up to 2048 processing elements (PE) [9, 17]. For the T3D, each PE contains a DEC Alpha 21064 microprocessor (clocked at 150 MHz) with an 8 KB instruction cache and an 8 KB data cache. One processing element node comprises two PEs and a network interface; the nodes are connected by a three-dimensional torus network. The network is composed of communication links and network routers which transfer packets through the communication links. The six communication links of each node are able to simultaneously support hardware transfer rates of 300 MB/s. The routers contain an X-dimension switch, a Y-dimension switch, and a Z-dimension switch which control the flow of packets through the different dimensions using the routing information of the packet. The X-dimension switch steers packets from one X-dimension communication link to the other or from one X-dimension communication link to the Y-dimension switch. The Y-dimension switch and the Z-dimension switch work identically. Although the memory is physically distributed, a logically shared memory is provided, i.e., any microprocessor can access data in the local memory of any PE without involving the microprocessor in that PE. Each virtual address is converted into a logical node number, PE number, and local address offset. If the PE number matches the PE number of the generating PE, a local memory access is performed. If the numbers do not match, a remote memory access is initiated by sending the PE number and the local address offset to the network interface. A remote memory read can be performed by the function \texttt{shmget()} from the Shared Memory Access (SMA) library [5]. This function copies a number of words from the local memory of a specified PE to the memory of the calling PE. A remote memory write can be performed using \texttt{shmmput()} which copies a number of words from the memory of the calling PE into the memory of a specified PE. None of these calls changes the entries of the data cache of the remote processor. The startup time for both calls lies between 1 and 2 \( \mu s \). The bandwidth for \texttt{shmget()} is 120 MB/s. The bandwidth for \texttt{shmmput()} is 60 MB/s because a remote load consists of a request followed by a remote write by the remote PE to the calling PE. The T3D provides special hardware support for fast barrier and eureka synchronizations. Both can be used if the entire user partition participates in the synchronization. Otherwise, software synchronization has to be used. The T3E uses an Alpha 21164 microprocessor clocked at 300 MHz, 450 MHz (T3E-900) or 600 MHz (T3E-1200) with an 8 KB direct-mapped instruction and data cache and an 96 KB three-way associative on-chip L2 cache. The 3D interconnection network provides a bidirectional bandwidth of 600 MB/s for each physical link. The access to remote data is performed via 640 E-registers. The latency for an access to local and remote memory is 283 ns or 1500 ns, respectively. The sustained bandwidth to local and remote memory is 630 MB/s and 300 MB/s, respectively.

6 The Performance Analysis

Based on these mathematical background described above, we will make the following assumptions suggested in [7, 8, 18] for our performance model. First, the model assumes perfect load balance and each processor holds a sufficiently large number of successive rows of the matrix, and the corresponding sections of the vectors involved. That is, our problems have a strong data locality. Secondly, we can compute the inner products (reduction) in two steps because the vectors are distributed over the processor topology. The computation of an inner product is executed by all processors simultaneously without communication or locally and these partial results are combined by a global sum operation and accumulated at a single destination.
processor, called single-node accumulation (SNA). The second phase consists of reversing the directions and sequence of messages sending the final results from this single processor to all other processors, called single-node broadcast (SNB).

In the following part we will describe a simple performance model including the computation time and communication cost for the main kernels as we presented before based on our assumptions. These two important terms are used in our paper suggested in [8]:

- Communication Cost: The term to indicate all the wall-clock time spent in communication, that is not overlapped with useful computation.

- Communication Time: The term to refer to the wall-clock time of the whole communication.

In the non-overlapped communication, the communication time and the communication cost are the same term.

6.1 Computation time

The IQMR algorithm contains three distinct computational tasks per iteration:

- Two simultaneous matrix-vector products, $A\vec{v}_n$ and $A^T\vec{w}_{n+1}$ whose computation time are given $2t_f/N/P$.

- Five simultaneous inner products, $(\vec{v}_{n+1}, \frac{\vec{v}_{n+1}}{n_{z}})$, $(\vec{w}_{n+1}, \frac{\vec{w}_{n+1}}{n_{z}})$, $(s_{n+1}, \frac{\vec{v}_{n+1}}{n_{z}})$ and $(r_{n-1}, \frac{r_{n-1}}{n_{z}})$ whose computation time are given by $(2n_{z} - 1)t_f/N/P$.

- Nine vector updates, $q_n, \vec{u}_n, \vec{v}_n, \vec{v}_{n+1}, p_n, -d_n, f_n, x_n$ and $r_n$ which are given $2t_f/N/P$.

where $N/P$ is the local number of unknown of a processor, $t_f$ is the average time for a double precision floating point operation and $n_{z}$ is the average number of non-zero elements per row of the matrix.

The complete (local) computation time for the IQMR method is given approximately by the following equation:

$$T_{comp}^{IQMR} = (19 + 2n_{z})\frac{N}{P}t_f = f(m)\frac{N}{P},$$

6.2 Communication cost

Before we compute the communication cost for the IQMR method, we need to know the communication cost of the single-node broadcast. For the single-node (accumulation or) broadcast operation which sends the same message from one processor to all other processors involved, the following variants are mainly considered in [16] for PVM:

Based on the study in [16], considering the execution time of the broadcast operation on 256 processors of the T3D and T3E for different messages sizes shows that for small message sizes of up to 4096/8192 byte, all variants except the $pvm_{psend}$ variant have nearly the same execution time. Figure 1 shows the results for the CrayT3E from [16]. If we look at the larger message sizes, the piecewise operations from their study show the best performance which means that it is better to split a large broadcast message into small pieces than to send it as a large message. It is also concluded from the study that the difference is more than an order of magnitude for messages of more than 1 Megabyte. This is usually not expected for the distributed memory systems. The explanation for this communication behavior can be found in [16]. A direct comparison between the $MPI\_Beast()$ operation and the (piecewise) $PVM\_broadcast$ operations shows that the MPI operations are faster than the PVM operations.

A comparison in [16] of the runtimes for maximum accumulation operations with PVM and MPI

![Figure 1. Results of runtime for single-node broadcast operations](image-url)
for 256 processors of the T3D with pvm_reduce and MpiReduce, respectively, shows that for small message sizes, the PVM operation is much faster than the MPI operation. For larger message sizes, both operations have about the same runtime, but the MPI operation shows much more fluctuations than the PVM operation. For messages of more than 8000 bytes, the difference is usually below 10% [16].

Now, we will show how the runtimes of the collective communication operations on the T3D and T3E can be modelled by using runtime formulas. Here note that runtime formulas depend on various machine parameters including the number of processors, the bandwidth of the interconnecting network, and startup times for the corresponding operations. We present the modelling results from [16] for our work by using the fastest variants of PVM and MPI communication operations, respectively. The runtime behavior of the other variants can be modelled similarly. The corresponding PVM and MPI operations can be modelled by the same runtime function with different coefficients. The runtime function for single-node broadcast or accumulation are summarized as follows:

\[ t_{str}(P, b, V) = \tau(V) \log_2 P + \tau_c(V)b \log_2 P, \]

where the value \(b\) is the message sizes in bytes, \(P\) is the number of processors and \(V\) is the specific variant of the communication operation. The specific values for the coefficients \(\tau(V)\), \(\tau_c(V)\) are given in Table 2 from [16]. The values result from curve fitting with the least-squares method. The logarithmic dependence on the number \(P\) of processors is used because the broadcast transmissions are based on broadcast trees with logarithmic depth. The same formula can also be used for the prediction of single-accumulation operations.

We use the results from [16] presented in Figure 2 to demonstrate the accuracy of the predicted and measured runtimes for PVM and MPI single-node accumulation operation. For the PVM broadcast, the piecewise variant with the InPlace option is used.

Finally, based on the single-node accumulation and broadcast time for 1 inner product, the communication time of the IQMR method is given as follows:

\[ T_{comm}^{IQMR} = 2\tau(V) \log_2 P + 2\tau_c(V)b \log_2 P. \]

6.3 Theoretical analysis

In this section, we will focus on the theoretical analysis of the parallel performance of IQMR method where the efficiency, speed-up and runtime are expressed as functions of the number of processors scaled by the number of processors that gives the minimal runtime for the given problem size.

The total runtime for IQMR is given by the following equation:

\[ T_p^{IQMR} = T_{comm}^{IQMR} + T_{comm}^{IQMR} = f(m)\frac{N}{P} + g(m) \]

This equation shows that for sufficiently large \(P\) the communication time will dominate the total runtime.

Let \(P_{max}\) denote as the number of processors that minimizes the total runtime \(T_p\) for any \(P\) processors of IQMR method, we can easily get

\[ P_{max} = \frac{f(m)N\ln 2}{2\tau(V) + 2\tau_c(V)b}. \]

7 Experimental Results

Here we mainly consider the partial differential equation taken from [1, 3, 4]

\[ Lu = f, \quad \Omega = (0, 1) \times (0, 1), \]

with Dirichlet boundary condition \(u = 0\) where

\[ Lu = -\Delta u - 20(\frac{\partial u}{\partial x} + y \frac{\partial u}{\partial y}), \]

and the right-hand side \(f\) is chosen so that the solution is

\[ u(x, y) = \frac{1}{2} \sin(4\pi x) \sin(6\pi y). \]

Basically, we discretize the above differential equation using second order centred differences on a 400 \(\times\) 400 with mesh size 1/441, leading to a system of 193600 linear equations with an unsymmetric coefficient matrix of 966240 nonzero entries. Diagonal preconditioning is used. For our numerical tests, we choose \(x_0 = 0\) as initial guess and \(tol = 10^{-5}\) as stopping parameter.

From the preliminary experimental results, the theoretical estimation gives very accurate prediction for the experimental results, the comparison shows that there is only 8% difference. Further detailed experimental results will be carried out for the comparison and presented later.

References


Table 2. Coefficient for runtime formula of single-broadcast [16]

<table>
<thead>
<tr>
<th>Message transfer variant</th>
<th>T3D</th>
<th>T3E</th>
</tr>
</thead>
<tbody>
<tr>
<td>Message transfer variant</td>
<td>$\tau(V)$</td>
<td>$t_s(V)$</td>
</tr>
<tr>
<td>PVM piecewise broadcast</td>
<td>53.65 $\mu$s</td>
<td>0.0130 $\mu$s</td>
</tr>
<tr>
<td>MPI broadcast</td>
<td>31.26 $\mu$s</td>
<td>0.0153 $\mu$s</td>
</tr>
<tr>
<td>PVM accumulation with $\max$</td>
<td>17.15 $\mu$s</td>
<td>0.036 $\mu$s</td>
</tr>
<tr>
<td>MPI accumulation with $\max$</td>
<td>87.10 $\mu$s</td>
<td>0.036 $\mu$s</td>
</tr>
</tbody>
</table>

Figure 2. Results of runtime for single-node broadcast operations


