Parallel-Processing Software
for the PANDA Computer

by
O. Trutner
Advisor - Dr. I. Bar-On

Technical Report #720
March 1992
Parallel-Processing Software
for the
PANDA Computer

Abstract
The PANDA computer is a parallel computer under development at the Technion, and is intended to provide super-computing capabilities at a relatively low cost. The PANDA's computing power is based on several powerful processor boards, working in parallel. The project presented in this paper provides the basic software layer for writing parallel programs on the PANDA. This includes a software library for inter-processor communication for the C and FORTRAN programming languages, a shell for executing parallel programs on the PANDA, a windowed front-end displayed on a graphic workstation running the X Window System and a simulation environment for running PANDA programs on a standard UNIX computer, where the PANDA is not available.

Description
The PANDA is a parallel computer containing several processor boards. These boards are manufactured by SKY Computers (US) and sold as powerful standalone processor add-ons, marketed under the name "SKYbolt". Each such board has its own processor, memory and input-output facilities, and thus functions as a standalone computer that is capable of running a single program on its own. SKY Computers also provides the basic software environment for a standalone SKYbolt board. This includes compilers for the C and FORTRAN programming languages, mathematic software library, and some other utilities. The PANDA has several such boards on a single bus, as well as a Sun workstation which serves as a host computer. In order to take advantage of the power of these processors for the completion of one computing task, some kind of communication is needed between the processors. The project described in this paper provides the basic software layer of inter-processor communication, by means of a software library, callable from C and FORTRAN programs running on the PANDA processors. This library is designed with the following goals in mind:

- Serve as the most elementary software layer that works on several processors in parallel. Based on this layer, higher layers (such as a parallel programming language) may be built.
- Allow data transfer between processors at the fastest rate allowed by hardware.
- Provide for synchronization between programs running on different processors.
- Simple interface for the programmer.

Also provided by this project is a shell for running a program on several processor boards. The shell is a program running on the host computer which is responsible for allocating free processors to run the program, loading the program on the allocated processors, synchronizing the programs to start running together, and detecting their termination. The shell is also responsible for running the windowed
front-end; on a graphic workstation running the X Window System, every processor participating in the program has a window opened for it on the workstation screen. This window handles all input-output for its processor, and provides other controls over the program running on it. Since all windows are simultaneously displayed on the same screen, the user may select the window(s) representing the processor(s) of interest, while other windows may be hidden or closed. This results in a comfortable and user-friendly environment for running parallel programs.

Since at the time of this writing, the PANDA is one of a kind, a simulation environment is provided. This environment allows simulation of running parallel PANDA programs on any UNIX computer. The simulation environment comprises a simulated parallel programming software library, a simulation shell, a windowed front-end similar to the one described above, and C/FORTRAN compilers of the local computer. Programs written for the PANDA may thus be compiled with the SKY compilers and run on the PANDA, or be compiled with the local compiler and run on the simulation environment. This allows users to write and debug PANDA programs without actually having a PANDA around.

The PANDA Computer

The PANDA comprises several SKYbolt processor boards mounted on a single VMEbus, together with a Sun workstation that serves as a host computer (see Fig. 1). Each SKYbolt board has one Intel i860 processor that serves as an arithmetic processor (AP), one Intel 80860 processor that serves as a system processor, 8 mega-bytes of internal memory, and 1/4 mega-bytes of fast memory serving as a data cache to enhance memory access rate of the AP (see Fig. 2). User programs run on the AP, while the SP provides operating system services for the program running on the AP. The board memory is mapped to the VMEbus address space.

![Figure 1 - Structure of the PANDA computer](image-url)
thus allowing one SKYbolt to access
the memory of another. The SP may be
used to transfer large amounts of data
from one SKYbolt to another by user
program control, without stopping the local
program, and without the knowledge of
the remote program. Data transfer rates
may rise as high as 20 mega-bytes per
second, which allows for parallel programs
with extensive communication. The Sun
workstation serves as a host computer;
it loads the programs onto the SKYbolts
and provides the with operating system
services that they cannot handle locally
(such as input-output, file system access,
etc.). All program development is done
on the Sun workstation (or on another
computer on the network). The SKYbolts
are used only for running the programs.

The Programming Model

Each PANDA processor is capable of
running an independent program in parallel
with the other processors. This program
starts its execution and terminates
without any effect on programs running
on other processors. However, since the
processors are all working together to
solve a single problem, the following
assumptions are made on the programming
model:

• All processors are executing the same
  program. They do not, of course, repeat
each other. Each processor determines
its part of the task by asking the parallel
library who it is.

• All processors start their work at the
  same time.

These assumptions are intended to

Figure 2 - Structure of a SKYbolt board
make the process of writing parallel programs simpler; Only one program has to be written, and there is no need to make sure that other processors have started running before communication with them is attempted. This programming model is assumed throughout the parallel programming library, and is enforced by the parallel execution shell.

The Parallel Programming Library

The parallel programming library provides the basic level of inter-processor communication. It is designed to both make maximum use of hardware ability, and to provide a simple interface to the programmer. The services it provides are divided into four categories:

- Initialization
- Synchronization - forcing processors to meet each other.
- Data transfer - moving data between processors.
- Cache management - to avoid cache coherency problems.

Listing 1
The library interface for the C programming language is shown in Listing 1. Detailed description of all library functions is found at the library manual pages, in appendix A.

Example Parallel Program

Listing 2 contains an example program that uses the parallel programming library, and may be run on the PANDA. It serves no practical purpose, except as an example for writing parallel programs. The program is loaded and run on a given number of processors, and is executed seperately on each. It first initializes the library using the par_init() call. All processors are numbered. The program then asks for how many processors there are, and what is the number of its own processor. This is done by calling par_get_num_procs() and par_who_am_i(), respectively. The program then attempts to meet the processor numbered next, by calling par_meet(). This would cause all processors to wait for their next, except for the processor numbered last. The last processor prints its number, and meets with its previous. The previous then prints its number too, and repeats the process. The result is that the processor numbers are printed in descending order, from last to first. For detailed description of the parallel functions used in this program, see the manual pages in appendix A.

Window Front-End to Parallel Programs

A PANDA program has many processors cooperating to solve a single problem. All processors may need to ask the user for input, and display results.

Listing 2

A single terminal would not suffice for developing such parallel programs, since the screen would be clobbered with output from different processors, and there would be no simple control of which processor gets the user input. When running on a workstation with a graphic display and a window system, this may be solved by opening a window for each processor. The parallel execution shell does just that if asked to, and the X Window system is running on the workstation. Each

```c
/*
   demo.c - an example program.
*/

#include<par.h>
main ()
{
    int ip, np ;

    /* Initialize */
    par_init () ;
    ip = par_who_am_i () ;
    np = par_get_num_procs () ;

    /* Meet higher processor, if any. */
    if (ip < np - 1)
        par_meet (ip + 1) ;

    /* Print who I am. */
    printf ("I am %d\n", ip) ;

    /* Meet lower processor, if any. */
    if (ip > 0)
        par_meet (ip - 1) ;
}
```
processor is assigned a window which displays output and sends back input when it has the input focus. Windows that display output of the processor(s) of interest may be brought to the foreground, while other windows may be closed to a small icon, and would not clobber the screen. Each window may be scrolled to display past output that was removed from the window as new output came. An example screen-shot of a workstation displaying output from one program on several windows is shown in fig 3.

Figure 3 - screen-shot of windowed front-end
Appendix A – Manual Pages
NAME
parexec – run a parallel program on the PANDA computer

SYNOPSIS
parexec [-n procs] [-w] program [...]

DESCRIPTION
parexec runs the specified PANDA program on the number of PANDA processors specified by the -n flag. The PANDA program is expected to be linked with the parallel library (libpar860.a), and to have a call to par_init() as its first statement. parexec guarantees that after each program returns from the call to par_init() all processors have loaded and started running the program.

OPTIONS
- n procs
The program is loaded and run simultaneously on procs processors. The default is one processor.

-w parexec opens an independent input/output window for each processor. This option is available only when the X Window System is running and set up correctly. The xtdout(1) program is used to implement the window semantics.

BUGS
In some cases of abnormal exit of a PANDA program, parexec dies while leaving some child processes around. These need to be killed manually before other PANDA programs may be run.

parexec allows only one PANDA program to execute at a time, even if there are unused processors which might have been used to run other programs.

SEE ALSO
parsim(1), xtdout(1), par_intro(3), par_init(3)

AUTHOR
Written on Mar 1 1992 by Oren Trutner at the IS lab in the Computer Science faculty at the Technion. If you have any problems with this routine, please contact oren@isaac.cs.technion.ac.il or baron@csacs.technion.ac.il

This software is provided "as is" without express or implied warranty.
NAME
parsim – run a simulation of the PANDA computer

SYNOPSIS
parsim [-n procs] [-w] program [...]

DESCRIPTION
parsim simulates running a program on the PANDA computer. The simulation runs on a Sun worksta-
tion. The program could be compiled by any Sun compiler, provided that it is linked with the simulation parallel library (libparsim.a), from which it must call par_init() as its first statement.

OPTIONS
-n procs
parsim simulates running the program on procs processors.

-w parsim opens an independent input/output window for each simulated processor. This option
is available only when the X Window System
is running and set up correctly. The xtdout(1)
program is used to implement the window semantics. parexec(1), xtdout(1), par_intro(3),
par_init(3)

AUTHOR
Written on Mar 1 1992 by Oren Trutner at the IS lab in the Computer Science faculty at the Technion.
If you have any problems with this routine, please contact oren@isaac.cs.technion.ac.il or
baron@csa.cs.technion.ac.il

This software is provided "as is" without express or implied warranty.
NAME
par_intro – introduction to the parallel library functions

DESCRIPTION
These functions constitute a basic protocol for inter-processor communication. The library is designed to produce optimal results on the PANDA computer. The PANDA computer has several SKYbolt processor boards and one Sun workstation mapped on one VMEbus. Each SKYbolt board has one Intel i860 arithmetic processor (AP), one Intel 80960 system processor (SP), at least 4 Mbytes of internal memory and a large data cache for the AP. User programs run on the AP, whereas the SP provides communication and other services. The internal memory is mapped to VMEbus address space, thus allowing access from other SKYbolt boards. The SP may be used to transfer large amounts of data from its AP to another SKYbolt, or vice versa, in the background, without disturbing the AP's on both sides of the transfer. The Sun workstation serves as host for the system. It downloads user programs onto the SKYbolt boards, and provides operating system services.

There also exists a software simulator for this environment. It is implemented on a standard Sun workstation, and uses processes to simulate AP processors. Source code compiled and run on the simulator may be recompiled for the SKYbolt and run on it. The simulator includes the parallel library.

The execution models under which the parallel library operates has all the SKYbolt boards running the same code, with the code started at the same point of time. Services provided by the library include identification of the program environment (number of processors, etc.), synchronization between processors, inter-processor data transfer and cache management.

FILES
libpar860.a parallel library for the Intel i860 processor
libsim.a parallel library for simulation on a Unix workstation

COPYRIGHTS
SKY and SKYbolt are copyrights (c) of SKY Computers, Inc., 1989, 1990, 1991. Intel, i860 and 80960 are trademarks of the Intel Corporation. Sun, Sun3, Sun4 and SPARCstation are trademarks of Sun Microsystems, Inc.

SEE ALSO

AUTHOR
Written on Mar 1 1992 by Oren Trutner at the IS lab in the Computer Science faculty at the Technion. If you have any problems with this routine, please contact oren@isaac.cs.technion.ac.il or baron@csa.cs.technion.ac.il

This software is provided "as is" without express or implied warranty.
NAME  
par_cache, par_uncache – setting cache access for memory regions

SYNOPSIS
#include <par.h>
void par_cache (addr, len)
   void *addr ;
   int len ;
void par_uncache (addr, len)
   void *addr ;
   int len ;
#define par_cache_var(var)  
   par_cache (&var, sizeof (var))
#define par_uncache_var(var)  
   par_uncache (&var, sizeof (var))

DESCRIPTION
par_cache() marks the memory region starting at address addr with length len as allowable for caching. This gives faster responses on repeated access to this region, but disallows any data transfer to/from it. Note that calling par_cache() does not imply moving any part of main memory into the data cache.

par_uncache() marks the memory region starting at address addr with length len as forbidden for caching. This gives slower responses on repeated access to this region, but allows data transfer to/from it. Note that calling par_uncache() does not imply flushing any part of the data cache.

By default all memory is marked as cacheable.

par_cache_var() and par_uncache_var() are convenience macros which remove the need to specify size for a simple variable.

SEE ALSO
par_intro(3), par_init(3), par_flush(3), par_get(3), par_put(3),

AUTHOR
Written on Mar 1 1992 by Oren Trutner at the IS lab in the Computer Science faculty at the Technion. If you have any problems with this routine, please contact oren@isaac.cs.technion.ac.il or baron@csa.cs.technion.ac.il

This software is provided "as is" without express or implied warranty.
NAME
par_flush — flush part of data cache into main memory

SYNOPSIS
#include <par.h>

void par_flush (addr, len)
    void *addr ;
    int len ;

#define par_flush_var(var,len) \
    par_flush (&var, sizeof (var))

DESCRIPTION
par_flush() flushes all references in the data cache to addresses between addr and addr+len to main memory. This qualifies that memory region to serve as source for a data transfer.

par_flush_var() is a convenience macro that removes the need to specify size for a simple variable.

BUGS
For the Intel i860 processor, len is internally rounded up to the nearest multiple of 4096.

SEE ALSO
par_intro(3), par_init(3), par_uncache(3), par_cache(3), par_get(3), par_put(3),

AUTHOR
Written on Mar 1 1992 by Oren Trubler at the IS lab in the Computer Science faculty at the Technion. If you have any problems with this routine, please contact oren@isaac.cs.technion.ac.il or baron@csa.cs.technion.ac.il

This software is provided "as is" without express or implied warranty.
NAME
par_get – read data from memory of another processor

SYNOPSIS
#include <par.h>

void par_get (proc, src, dst, len)

int id ;
void *src ;
void *dst ;
int len ;

#define par_get_var(proc,src,dst) \n     par_get (proc, &src, &dst, sizeof (dst))

DESCRIPTION
par_get() copies len bytes starting at address src in the memory space of processor proc, to address dst in the memory space of the calling processor. par_get() returns immediately, without waiting for the data transfer to complete, thus allowing a large data transfer to execute in the background, while the program continues to run. To check whether the transfer has finished, use par_transfer_done().
par_get() does not disturb the program running on processor proc, which is completely unaware of the data transfer from its memory space.

Note that the transfer is done from and to main memory, disregarding any data cache. This means that the memory area used as destination must be uncached (see par_uncache() ), and that the memory area used as source must be either uncached or have all cache references to it flushed just before the transfer start (see par_flush() ).

For fastest results on current hardware, both src and dst must have exactly the same alignment to a 16-byte boundary. Since src refers to an address on a remote processor, if an address of a local variable is passed as src, that variable must either be global, or declared as static.

par_get_var() is a convenience macro, which is useful for cases when both the source and the destination of the transfer are variables. It removes the need to reference the addresses and the size of these variables.

SEE ALSO
par_intro(3), par_init(3), par_flush(3), par_uncache(3), par_put(3), par_transfer_done(3)

AUTHOR
Written on Mar 1 1992 by Oren Trutner at the IS lab in the Computer Science faculty at the Technion. If you have any problems with this routine, please contact oren@isaac.cs.technion.ac.il or baron@csa.cs.technion.ac.il

This software is provided "as is" without express or implied warranty.
NAME
    par_get_num_procs – return number of processors on which the current program is running.

SYNOPSIS
    #include <par.h>
    int par_get_num_procs()

DESCRIPTION
    par_get_num_procs() returns the number of processors on which the current program is running. This may be any number between 1 and the number of physical processors actually available.

SEE ALSO
    par_intro(3), par_init(3), par_who_am_i(3),

AUTHOR
    Written on Mar 1 1992 by Oren Trutner at the IS lab in the Computer Science faculty at the Technion. If you have any problems with this routine, please contact oren@isaac.cs.technion.ac.il or baron@csa.cs.technion.ac.il

    This software is provided "as is" without express or implied warranty.
NAME
par_init — initialize the parallel library

SYNOPSIS
#include <par.h>

void par_init()

DESCRIPTION
par_init() initializes the parallel library. It must be called before any other parallel library function, and most preferably, before any other statement in the program. The result of any statement executed before the call to par_init() is unpredictable.

SEE ALSO
par_intro(3), par_get_num_procs(3), par_who_am_i(3),

AUTHOR
Written on Mar 1 1992 by Oren Trutner at the IS lab in the Computer Science faculty at the Technion. If you have any problems with this routine, please contact oren@isaac.cs.technion.ac.il or baron@csa.cs.technion.ac.il

This software is provided "as is" without express or implied warranty.
NAME
par_meet - synchronize this processor with another processor

SYNOPSIS
#include <par.h>

void par_meet (proc)
int proc ;

DESCRIPTION
par_meet() hangs the calling processor until processor proc calls either par_meet() with the id of this processor as argument, or par_meet_any(). If processor proc is already hung either on par_meet() with the id of this processor as an argument, or on par_meet_any() when this processor calls par_meet(), then par_meet() returns immediately, and also releases the call on which the processor proc is hung.

SEE ALSO
par_intro(3), par_init(3), par_meet_any(3), par_meet_all(3),

AUTHOR
Written on Mar 1 1992 by Oren Trubler at the IS lab in the Computer Science faculty at the Technion. If you have any problems with this routine, please contact oren@isaac.cs.technion.ac.il or baron@csa.cs.technion.ac.il

This software is provided "as is" without express or implied warranty.
NAME
par_meecall – synchronize all processors

SYNOPSIS
#include <par.h>

void par_meecall()

DESCRIPTION
par_meecall() hangs the calling processor until all other processors have called par_meecall(). If all other processors are already hung on par_meecall(), then all processors are immediately released to continue execution. If some other processor is hung on anything other than par_meecall(), i.e. par_meet() or par_meecany(), then a deadlock condition occurs.

SEE ALSO
par_intro(3), par_init(3), par_meet(3), par_meecany(3)

AUTHOR
Written on Mar 1 1992 by Oren Trutner at the IS lab in the Computer Science faculty at the Technion. If you have any problems with this routine, please contact oren@isaac.cs.technion.ac.il or baron@csa.cs.technion.ac.il

This software is provided "as is" without express or implied warranty.
NAME
par_meet_any - synchronize this processor with some other processor

SYNOPSIS
#include <par.h>
int par_meet_any()

DESCRIPTION
par_meet_any() hangs the calling processor until some other processor calls either par_meet() with
the id of this processor as argument, or par_meet_any(). If some other processor is already hung either
on par_meet() with the id of this processor as an argument, or on par_meet_any() when this proces­
sor calls par_meet_any(), then par_meet_any() returns immediately, and also releases the call on
which the other processor is hung.

RETURN VALUES
par_meet_any() returns the id of the processor with which it has met.

SEE ALSO
par_intro(3), par_init(3), par_meet(3), par_meet_all(3),

AUTHOR
Written on Mar 1 1992 by Oren Trutner at the IS lab in the Computer Science faculty at the Technion.
If you have any problems with this routine, please contact oren@isaac.cs.technion.ac.il or
baron@csa.cs.technion.ac.il

This software is provided "as is" without express or implied warranty.
NAME
par_put - write data into memory of another processor

SYNOPSIS
#include <par.h>

void par_put (proc, dst, src, len)
  int id ;
  void *dst ;
  void *src ;
  int len ;

#define par_put_var(proc,dst,src) \
  par_put (proc, &dst, &src, sizeof (dst))

DESCRIPTION
par_put() copies len bytes starting at address src in the memory space of the calling processor, to address dst in the memory space of processor proc. par_put() returns immediately, without waiting for the data transfer to complete, thus allowing a large data transfer to execute in the background, while the program continues to run. To check whether the transfer has finished, use par_transfer_done(). par_put() does not disturb the program running on processor proc, which is completely unaware of the data transfer into its memory space.

Note that the transfer is done from and to main memory, disregarding any data cache. This means that the memory area used as destination must be uncached (see par_uncache() ), and that the memory area used as source must be either uncached or have all cache references to it flushed just before the transfer start (see par_flush() ).

For fastest results on current hardware, both src and dst must have exactly the same alignment to a 16-byte boundary. Since dst refers to an address on a remote processor, if an address of a local variable is passed as dst, that variable must either be global, or declared as static.

par_put_var() is a convenience macro, which is useful for cases when both the source and the destination of the transfer are variables. It removes the need to reference the addresses and the size of these variables.

SEE ALSO
par_intro(3), par_init(3), par_flush(3), par_uncache(3), par_get(3), par_transfer_done(3)

AUTHOR
Written on Mar 1 1992 by Oren Trutner at the IS lab in the Computer Science faculty at the Technion. If you have any problems with this routine, please contact oren@isaac.cs.technion.ac.il or baron@csa.cs.technion.ac.il

This software is provided "as is" without express or implied warranty.
NAME
  par_transfer_done – check whether inter-processor data transfer is in progress

SYNOPSIS
  #include <par.h>
  int par_transfer_done ()

DESCRIPTION
  par_transfer_done() returns non-zero iff no background data transfer initiated by par_put() or
  par_get() is in progress. May be used for determining if a previously initiated transfer has finished.

SEE ALSO
  par_intro(3), par_init(3), par_get(3), par_put(3),

AUTHOR
  Written on Mar 1 1992 by Oren Trutner at the IS lab in the Computer Science faculty at the Technion.
  If you have any problems with this routine, please contact oren@isaac.cs.technion.ac.il or
  baron@csa.cs.technion.ac.il

  This software is provided "as is" without express or implied warranty.
C LIBRARY FUNCTIONS

NAME
par_who_am_i - return id of calling processor

SYNOPSIS
#include <par.h>
int par_who_am_i();

DESCRIPTION
Each processor participating in the execution of a program using the parallel library is given an integer
identifier between 0 and par_get_num_procs()-1. par_who_am_i() returns the id of the calling processor,
thus allowing the program to find out which processor it is running on.

SEE ALSO
par_intro(3), par_init(3), par_get_num_procs(3).

AUTHOR
Written on Mar 1, 1992 by Oren Trutner at the IS lab in the Computer Science faculty at the Technion.
If you have any problems with this routine, please contact oren@isaac.cs.technion.ac.il or
baron@csa.cs.technion.ac.il

This software is provided "as is" without express or implied warranty.
Appendix B – Copyrights


Intel, i860 and 80860 are trademarks of Intel Corporation.

UNIX is a registered trademark of AT&T.

Sun, Sun Microsystems and Sun Workstation are registered trademarks of Sun Microsystems, Inc.