Measurements of MasPar MP-1216A Communication Operations

Lutz Prechelt (prechelt@ira.uka.de)
Institut für Programmstrukturen und Datenorganisation
Fakultät für Informatik
Universität Karlsruhe, Postfach 6980
D-7500 Karlsruhe, Germany
++49/721/608-4068, Fax: ++49/721/694092

January 6, 1993

Technical Report 01/93

Abstract

The MasPar MP-1 is a SIMD parallel computer with high throughput on fine-grain irregular interprocessor communication. This report presents measurements of communication on a MP-1216A machine with 16384 processors. The timings cover all classes of communication operations provided in the standard MPL library plus the router and xnet statements with a variety of communication and processor activity patterns. This report also discusses the results of these measurements, some of which are rather surprising.

Contents

List of Figures

1 Introduction

This section discusses why the measurements have been performed and for whom they are useful, what has been measured and what not, how the measurements have been done and how valid I expect them to be, how to interpret the diagrams. It ends with a short discussion and summary of results.

1.1 Why measure

The driving force behind these measurements was the wish to have data on which to base code
generation decisions for an optimizing compiler we are building for the MasPar MP-1. There
are cases in which a compiler can implement the same functionality in more than one way
it is not clear which way is most efficient. This is especially true when using librar
since their runtime behavior under different conditions is less transpare
communication operations — the MasPar library documentation doe
this matter. Such knowledge is not only useful for a compil
application programmer.

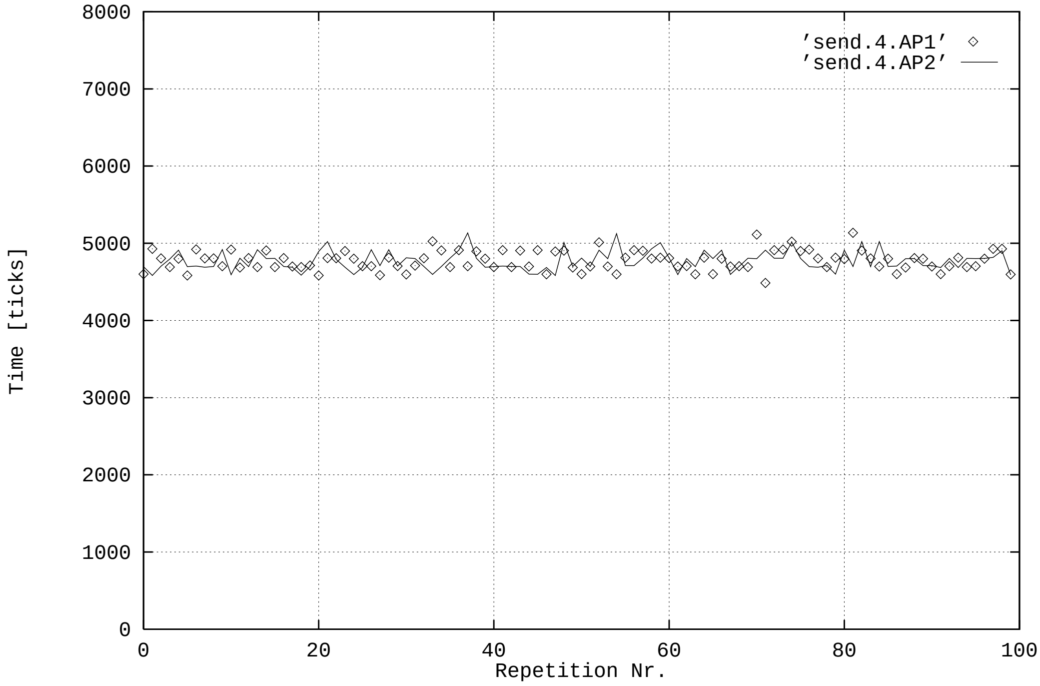
1.2 What was measured and how

The measurements cover only communication operations, fo

1. the cost of other basic operations can mostly be c
MasPar manuals.
2. often the most important implementation
tion operations.

For the rest of this report, I will ass
language and the MPL library.

All experiments have been done
been determined with the d
8 times with the same par
Exceptions to t
gathered
done onl
the



Variations in 100 trials for 4 byte router communication of a permutation. Points are for the 100 trials with a single random permutation that modified itself from trial to trial and the line is for trials with 100 diffe random permutations.

Figure 1: **router permutation sensitivity**

each trial. The results can be seen in Figure 1; runtimes vary by less than 10 perce behave very similarly for newly generated as for self-modified permutations.[1]

1.3 How to read the diagrams

Most diagrams indicate time on the ordinate (y-axis), measured in so-calle cycle of the internal clock on the MasPar DPU and lasts 80ns (i.e. Note, although many diagrams share the same scale on the vert

The abscissa (x-axis) of most diagrams is a logarithmic example a value of 12 means that 2^{12} PEs are in th three out of four processors are inactive.

For each curve, there is a name in t usually consists of three compo `OP` indicates the operatio name, for raw rout "fetch" indi `SZ` me a s probabilistic activity (each PE is individually active or inactive with a certain probability), or B for block (exactly the first n PEs are active). This first letter is irrelevant —and thus left ou —for xnet communication.

The second letter can be either P for a random permutation of destination PEs (each PE c exactly once in the pattern when all PEs are active), R for a random destinatio by each PE individually), C for cycling destinations (PEs with numbers di destinations with numbers differing by n modulo the number of destina (only n destinations allocated in a contiguous block are use destination", or S for shift (this letter is followed shift: 1 or 100).

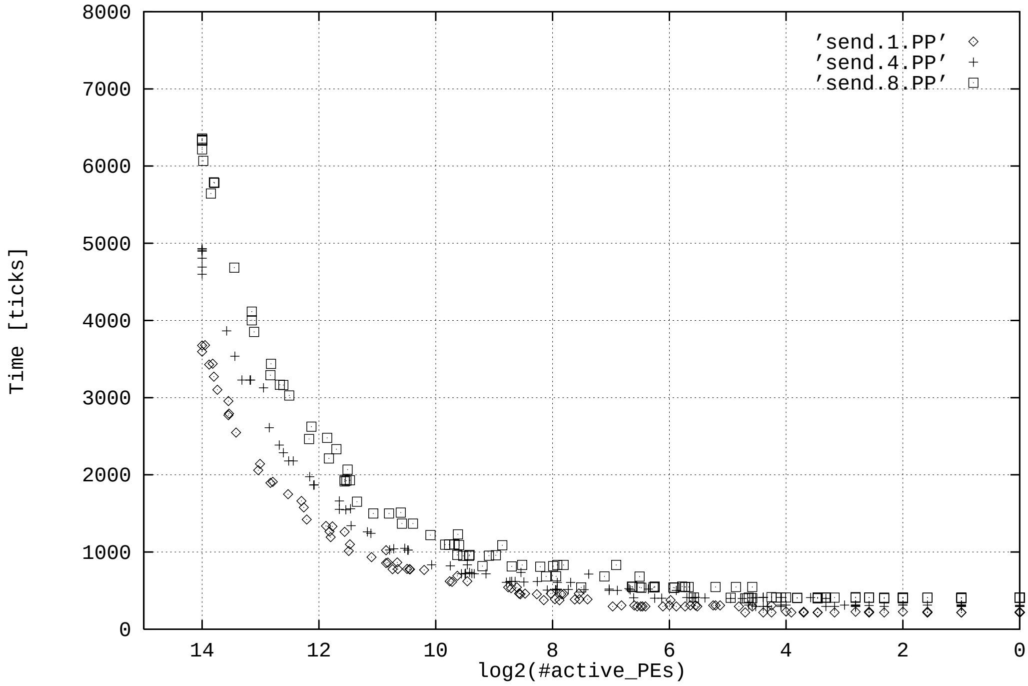
There are a few exceptions to these rules, figures. The communication patter up their own communicatio

The example fetch.4.PP activity pattern an set looks appr

```
openfil
for (j
  for (
    de
    if

  }
}
```

where j c mu



Left-hand-side usage (i.e. as lvalue) of a router statement in assignments of `char/integer/double`, respectively. The activity pattern is probabilistic, i.e. each PE is individually active or inactive with probability 2^{x-} communication pattern is a random permutation. Note that the highest value on the x-axis is on th

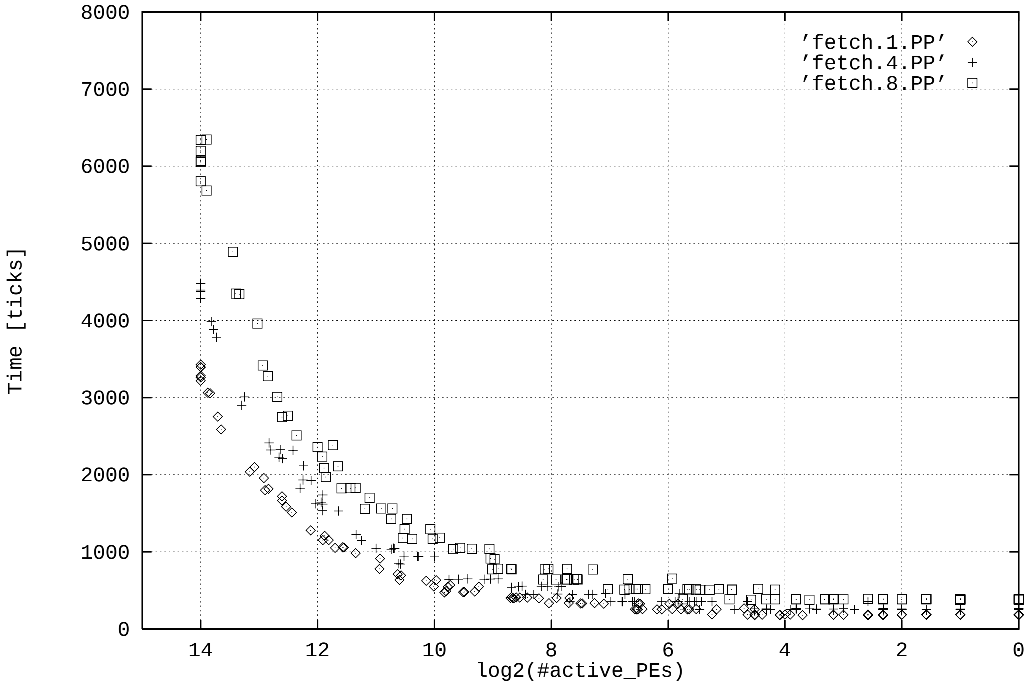
Figure 2: **router send**

2 Raw router communication

In this section we will examine the behavior of the router statement and the rsend and rfetch library functions.

2.1 The router statement

The only timing data the MasPar manuals provide on the cost of router communication is that
it communication will on the average take about 5000 ticks, if all PEs participate.[2] In
s a little smaller. This stems from the fact that a permutation is a
m pattern, because in a permutation each
cation with



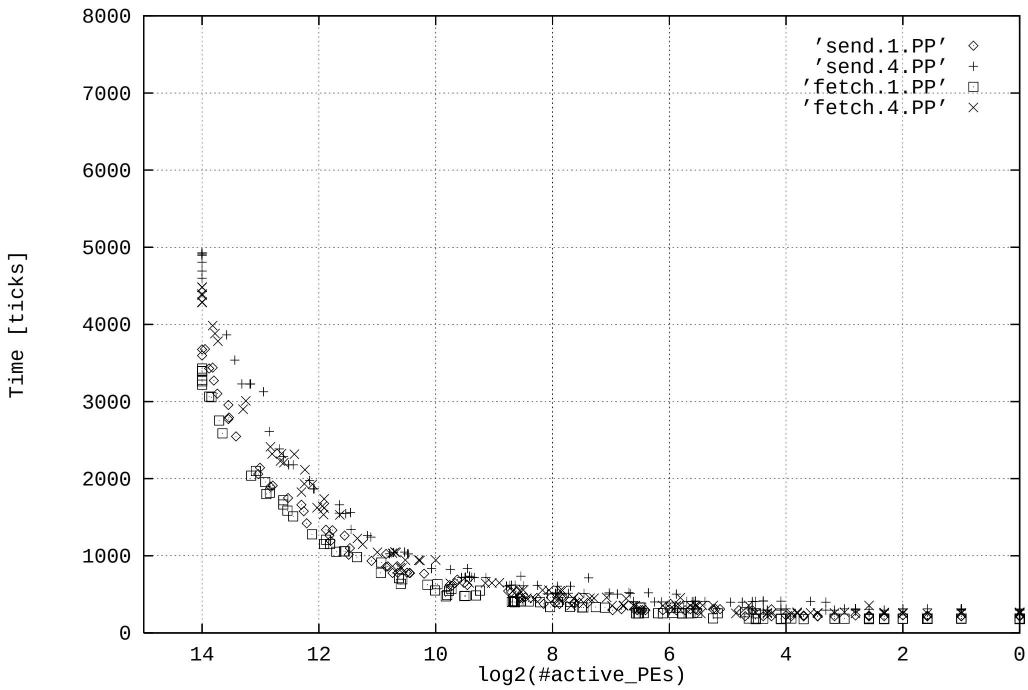
Right-hand-side usage of a router statement in assignments of char/integer/double, respecti
activity pattern and random permutation communication pa

Figure 3: **rout**

Note that for both, send and fetch, the cost of a router statment
in the communication.

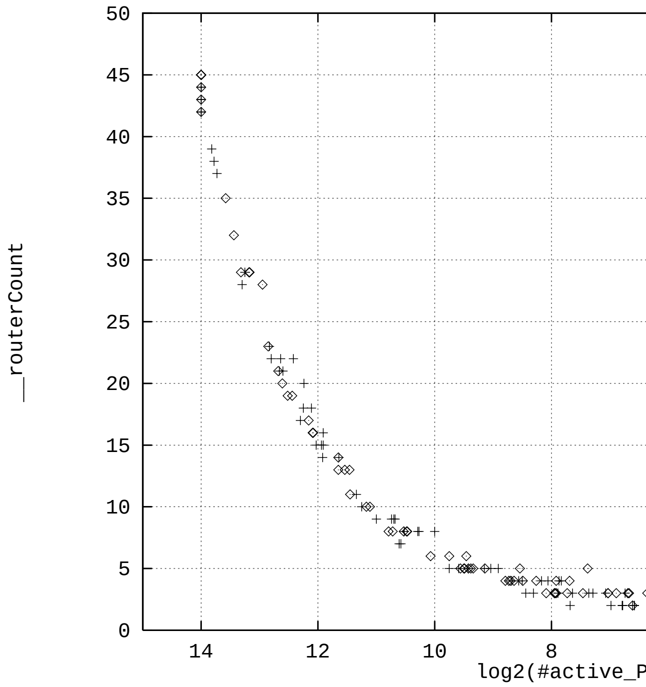
A direct comparison between send and fetch, as in figure 4, reveals that both operations indeed
show the same behavior in cost reduction as a reaction of activity reduction. Trying to charac-
terize this quantitatively we find the following rules of thumb in the diagram: communication
e PEs participating takes about 1/3 of the time that a communication of all PEs
5 time and with 1/64 of the PEs it takes 1/8 time.

a little more expensive than
s

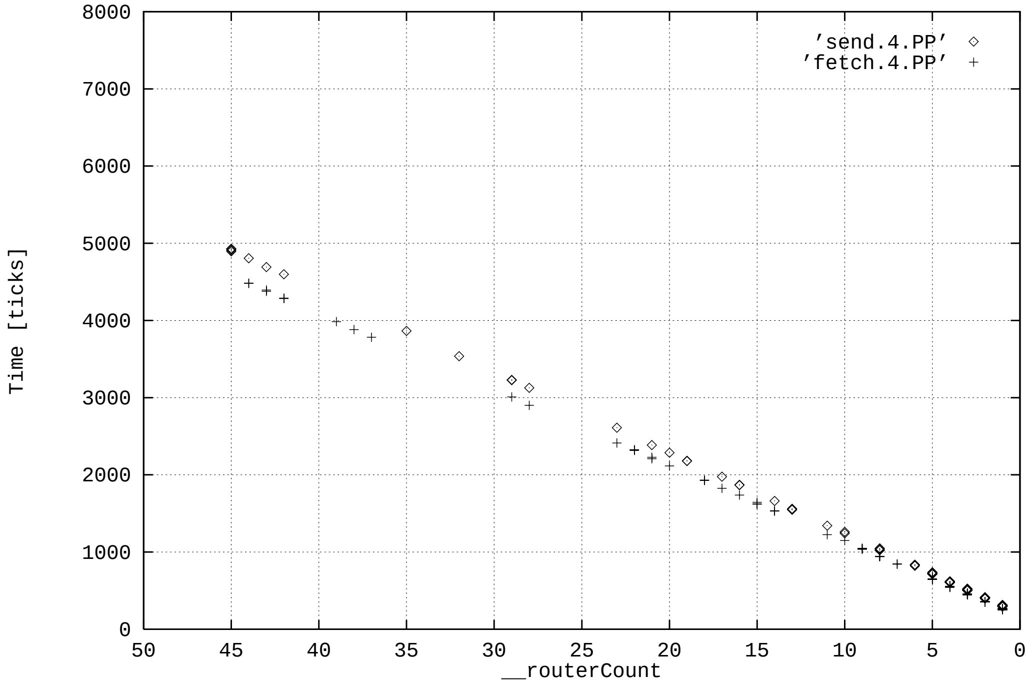


Left-hand-side usage vs. right-hand-side usage of a router statement in as

Figure 4: **router send vs**



Number of sequential communication steps ne
sponds



Time needed for a send or fetch depending on the number of sequential communication step
Each data point directly corresponds to one of figure

Figure 6: **router s**

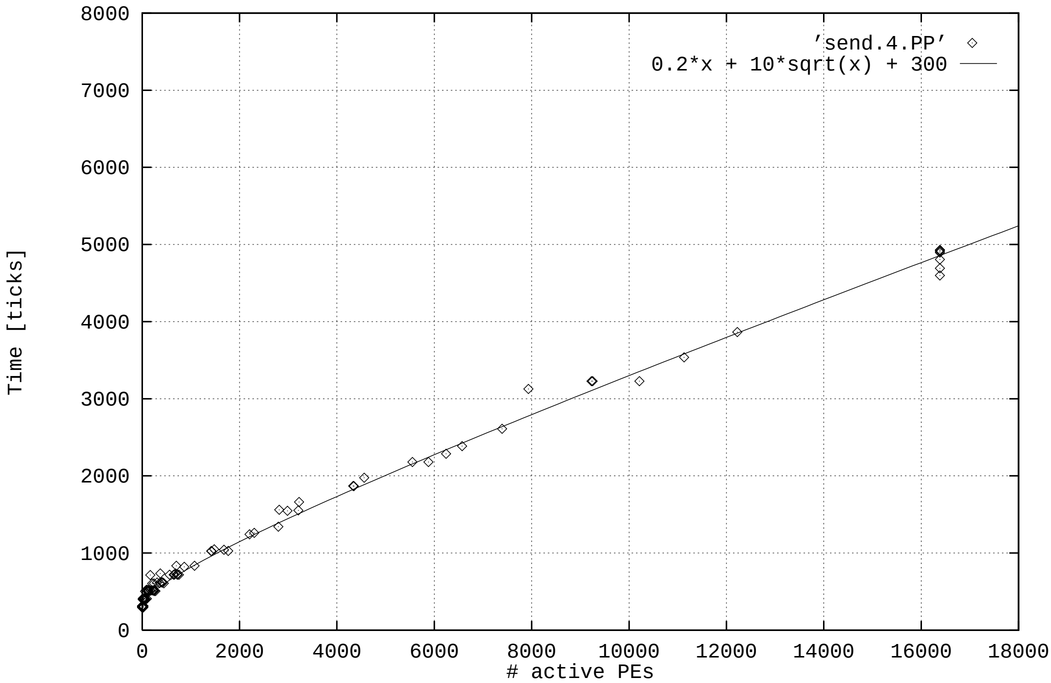
Thus, fetch is indeed a bit cheaper than send, which is ni
fetch tends to be used more often.

The above computations and the routerCount diagrams suggest an almost linear behavior of
router runtime with increasing absolute number of active PEs. By hand-fitting a function to
the 4-byte send curve, I found $x/5+10\sqrt{x}+300$ to give a good approximation (see figures 7
e behavior is not completely linear (at full activity the nonlinear part accounts
t is asymptotically linear.

erns

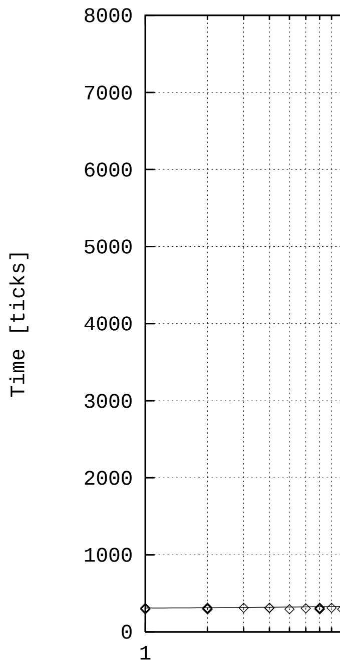
h difference whether the pattern of activity
difference, if the

2.3 Using other communication patterns

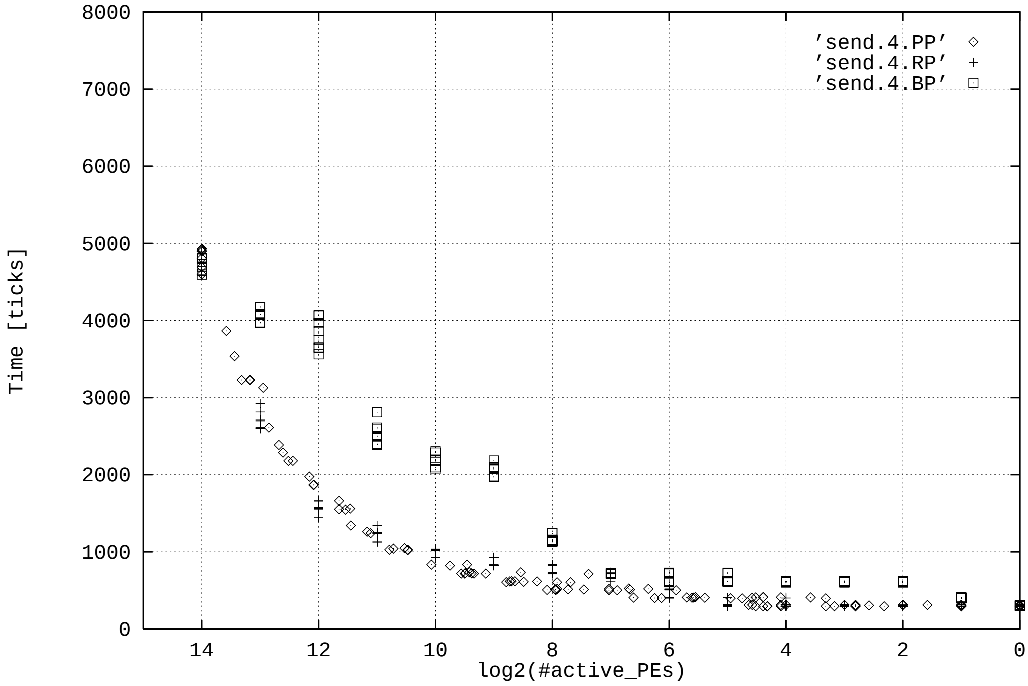


4-byte router send by PE activity and approximation function. Here a
highest value is on the right.

Figure



4-byte router send by PE



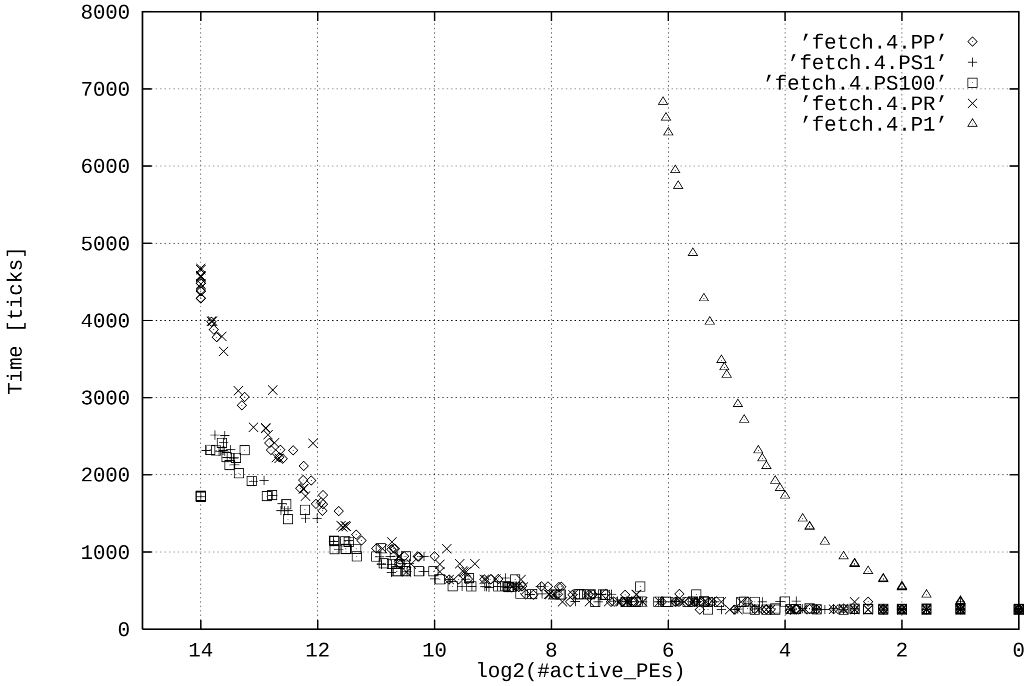
Left-hand-side usage of a router statement to communicate a random permutation with diffe
P is *probabilistic*, just like in the figures above, R is *reg*
2^{14-x} are active, an

The result

1. fetching from a s
with the number of active PEs.

2. Shifts (i.e. each processor i communicates with p
for all processors and n is the number of processors) are particular
on the MasPar router; for almost all PEs active their cost is only about 60 percent of
an average random permutation. If all PEs are active the cost drops further to about 50
percent of the standard value.

3. A random communication pattern is only slightly more expensive than a permutation. In a
h PE picks its partner independently of all other PEs, so collisions may
the MasPar router serializes into approximately 40
ditional cost. However,



Router fetch with probabilistic activation pattern and the following communi
(P), shift by 1 PE in iproc order (S1), s
duplica

2.4.

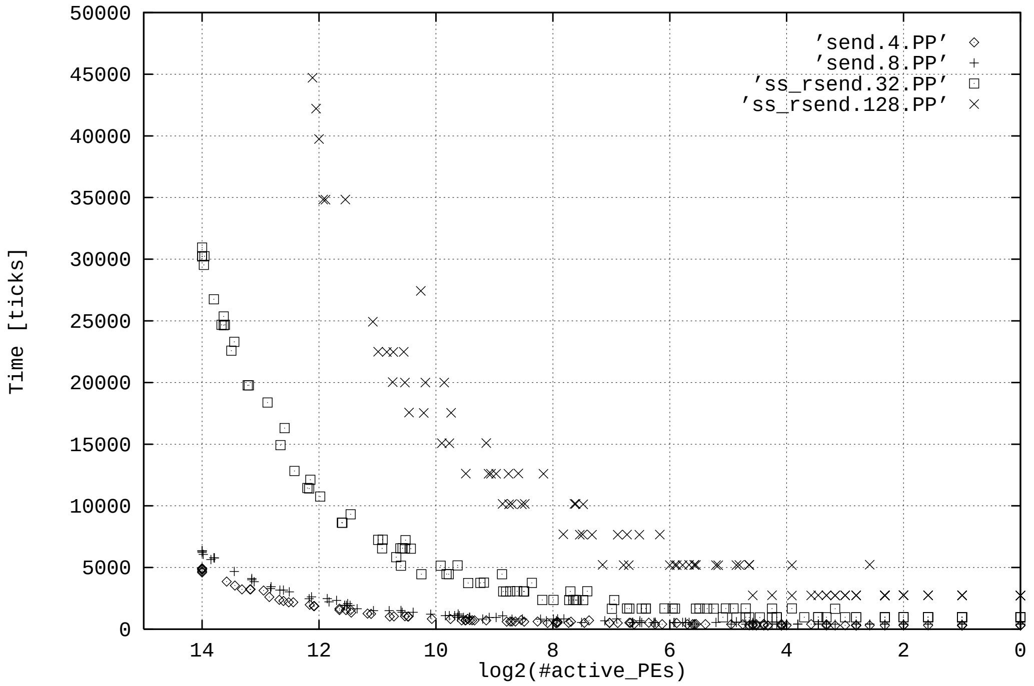
The tim
expect, communication time rises l
having fewer PEs active are similar to those of the bare r

2.4.2 rsend vs. rfetch

Comparing rsend and rfetch leads to a similar result as comparing send and fe
statements: Figure 13 indicates that both operations take about the same time, except when all
PEs are active where rsend takes a few percent longer than rfetch. This is true for singular as
well as plural pointers.

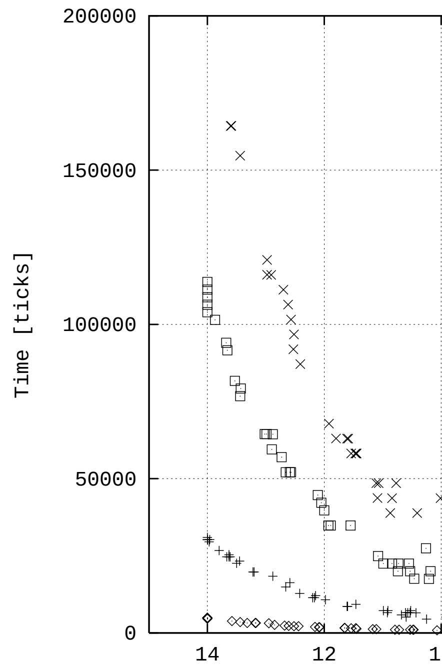
.4.3 rsend vs. router statement

d is more expensive than router on the same small amount of data. This is not surprising
rsend (being the more general command) to need a longer setup
r two for 4-byte packets. The lesson we

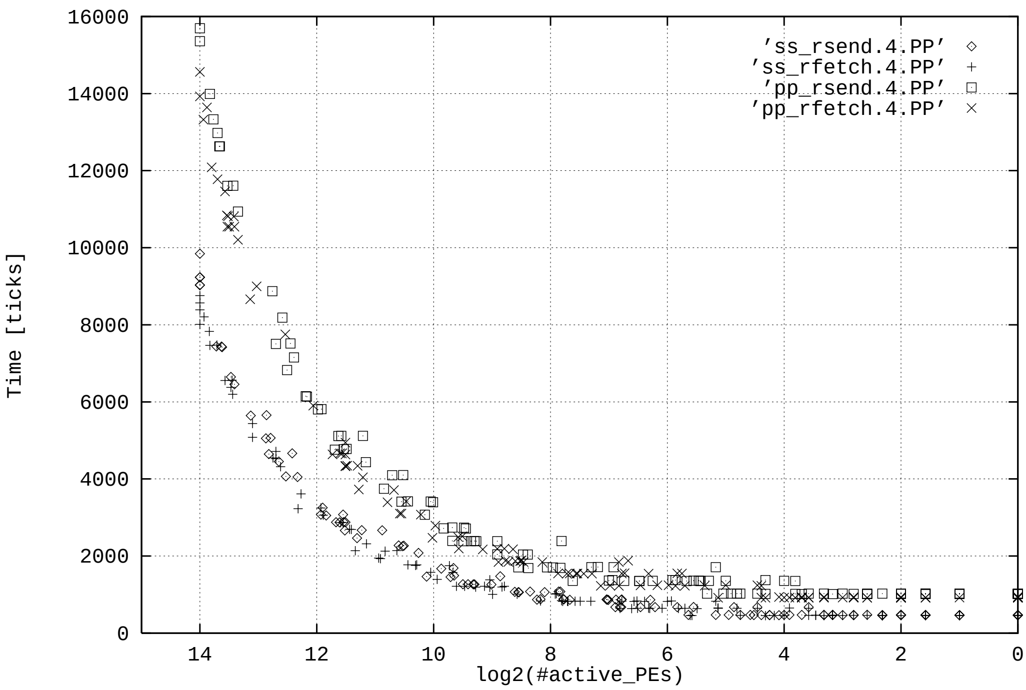


Using the router statement for communicating a permutation of 4 or 8 bytes, and ss_rsend
permutation of 32 or 128 bytes, respectively, with ranc

Figure 11:

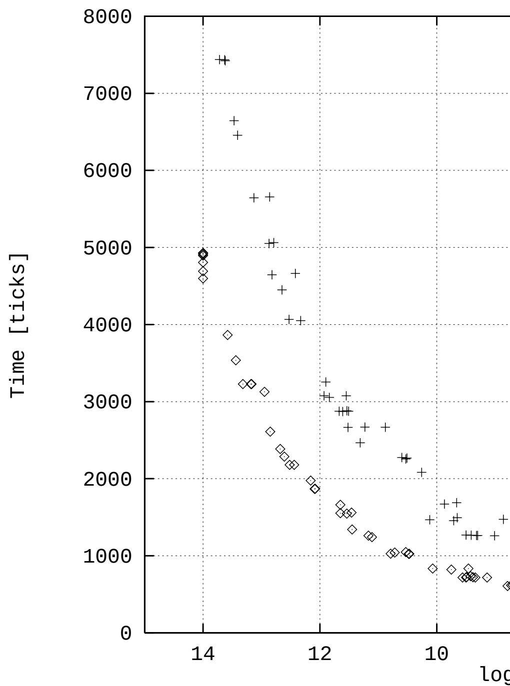


Using the router statement for

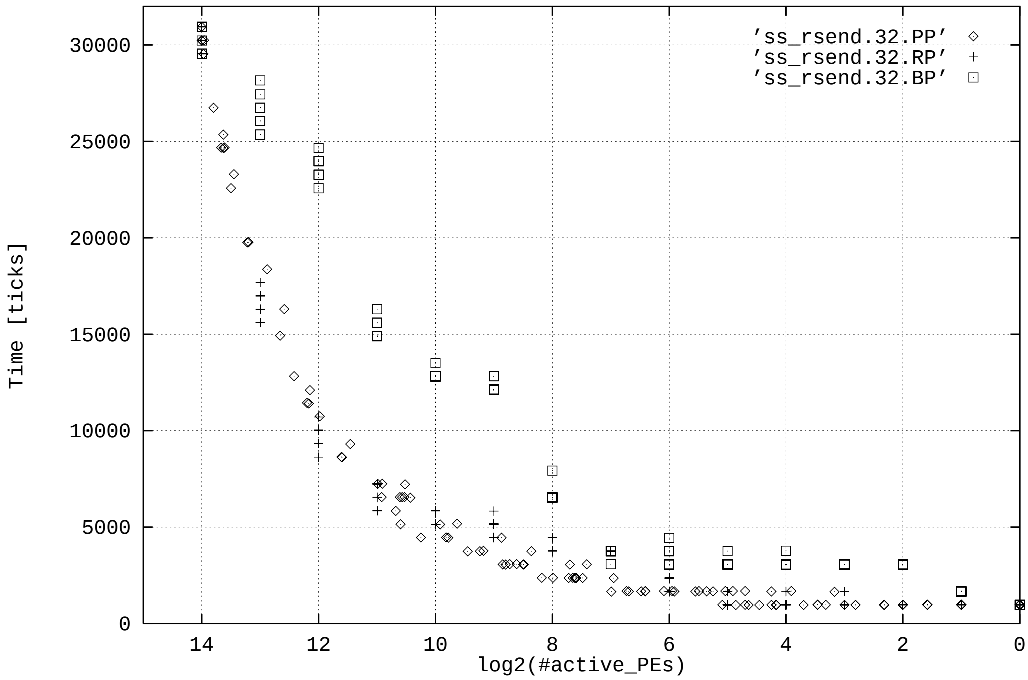


sending or fetching 4 bytes using all singular or all plural pointers, ra
permutation.

Figure 13: r



Left-hand-side usage of a router sta



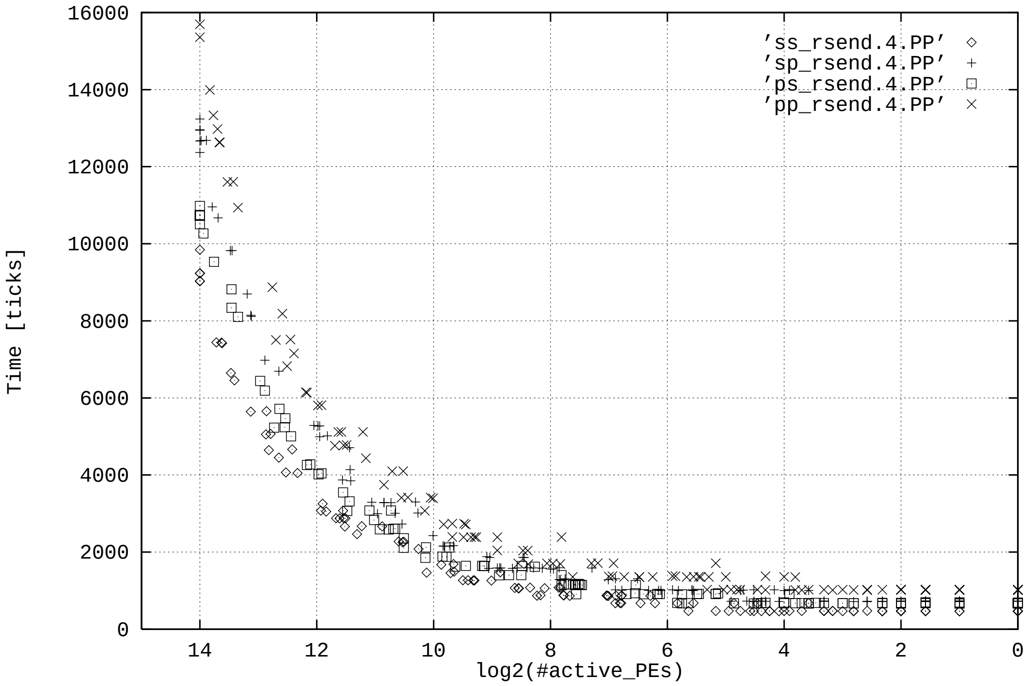
Corresponds to figure 9.

Figure 15: **rsend with probabilistic vs. regular activity**

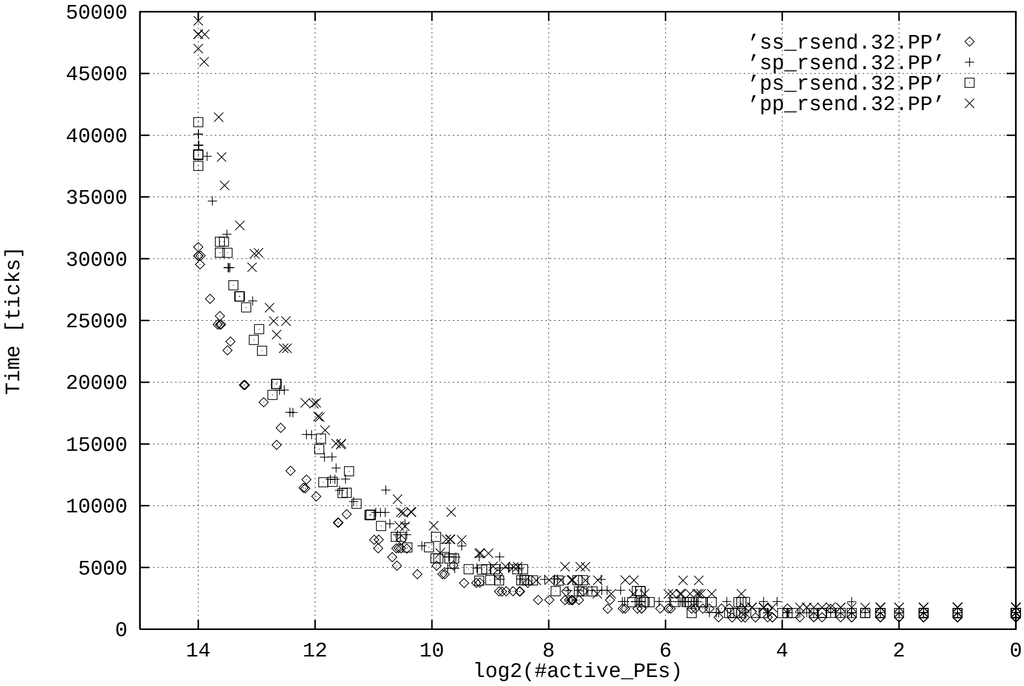
Using different activation patterns on rsend (figure 15) has the same effect as for the router
tement (see figure 9 and the discussion in section 2.2).

lar vs. plural pointers

iants of the rsend and the rfetch library functions: all four combinations of
e and destination addresses of the data to be transmitted
antly different cost for a small
s_rsend,



rsend of a 4-byte packet with singular source and singular destination dat
plural destination, with plural sour



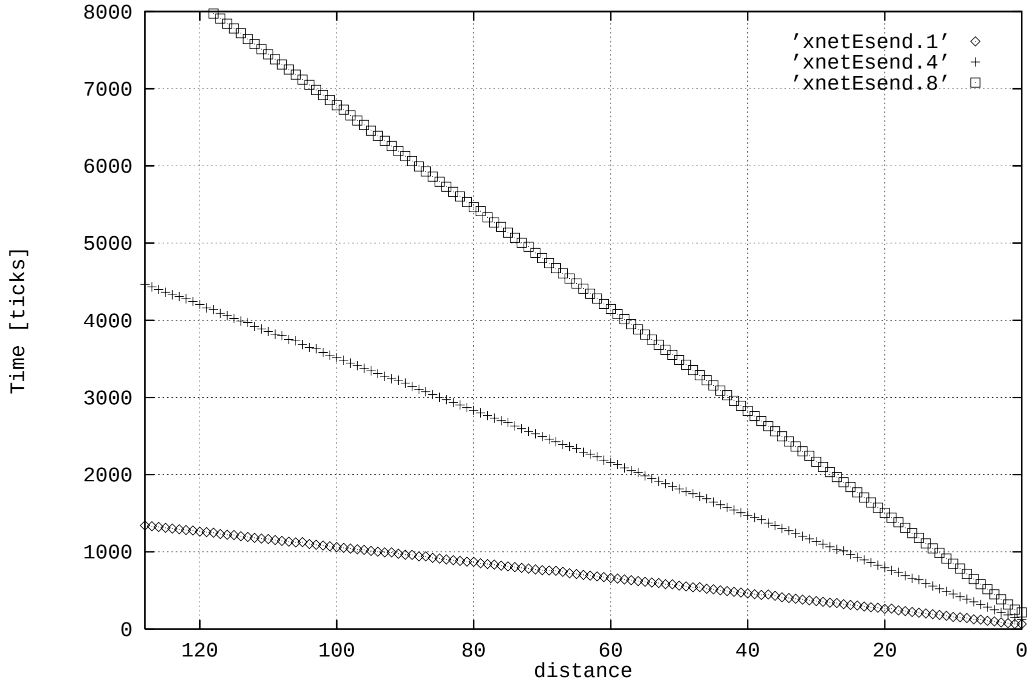
rsend of a 4-byte packet for larger packets with singular source and singular destina
singular source and plural destination, with plu
plural de

3 x

xnet
behavior is much more straightfor

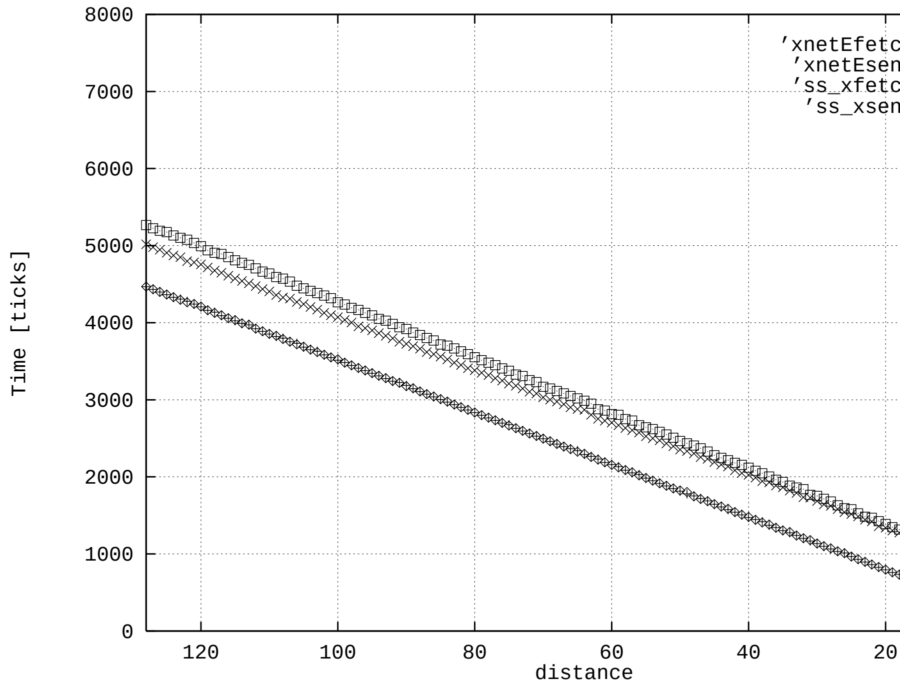
The times for raw xnet communication can be
manuals. I did some experiments with it anyway, in order to visualize the b
the formulae.

The first experiment is depicted in figure 19: xnet usage on the left hand side of an assignment
uses time proportional to the size of the data object and proportional to the communication
distance. This is true except for a small additional constant time and independent of the
E, N, NW, W, SW, S, or SE). The curves show almost no aberrations and
hough not perfectly: for the 4-byte send
nt gave 4206

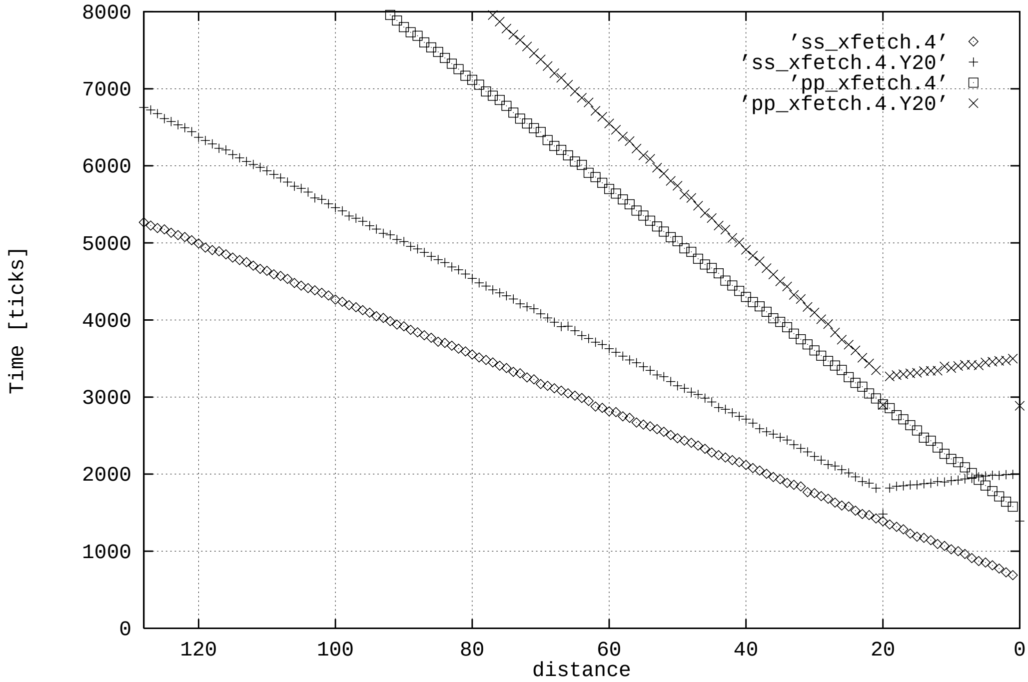


Left-hand-side usage of a xnet statement in assignments of char/integer/double, respecti

Figure 19: **xnet send**



sending versus fetching via xnet using the xnet statement c
for a 4-byte packet.



xfetch in standard and nonstandard directions: ss_xfetch in direction (x, 0), ss_xfetch
corresponding two curves for pp_xfetch.

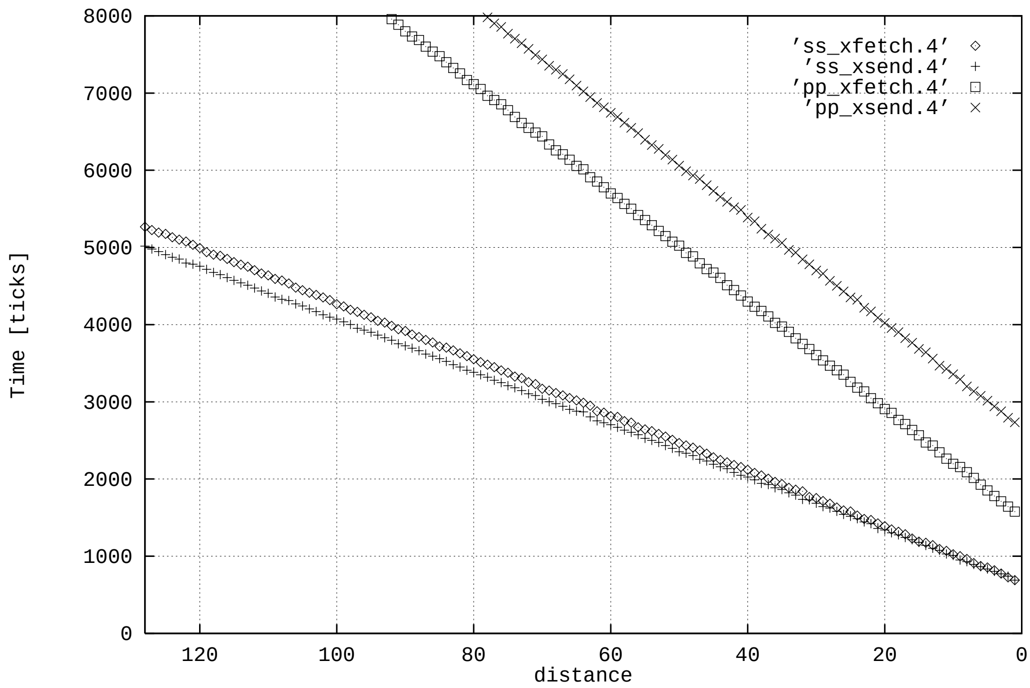
Figure 21: **xfetch**

ss_xfetch were used, as shown in the upper two curves of fi
than raw xnet send by a constant of about 700 ticks; the next experiment gives an i

The xsend and xfetch functions allow to use arbitrary offsets in x- and y-direction. While in
figure 20 only one direction was used and the other direction's distance was chosen as 0, we now
one direction constantly to distance 20 and graph the runtime (figure 21). The results are

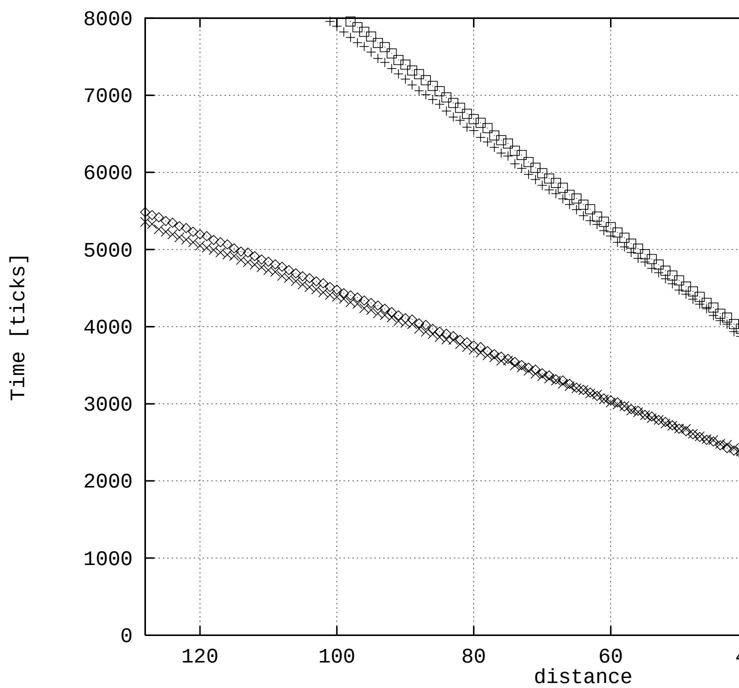
for $x < 20$ but even sinks for growing x.
re steeper than those for standard directions (E,

ard direction

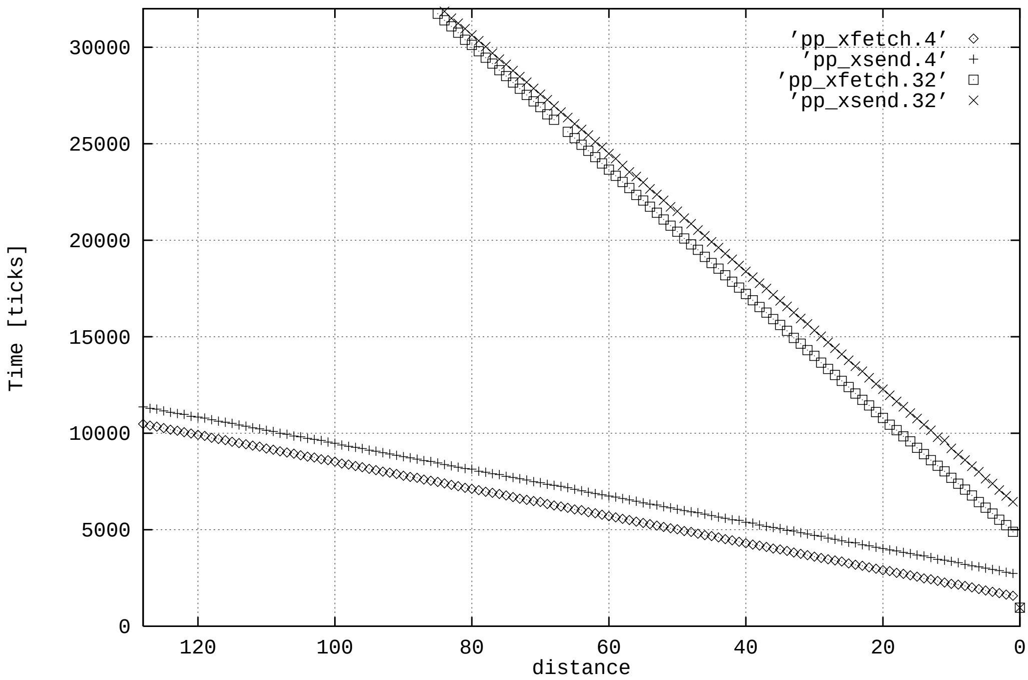


xfetch and xsend with singular source and destination address and with plural sourc

Figure 22: **ss_xfetch/send vs. pp_xfe**



xfetch and xsend with singular source and plural d
nation addr

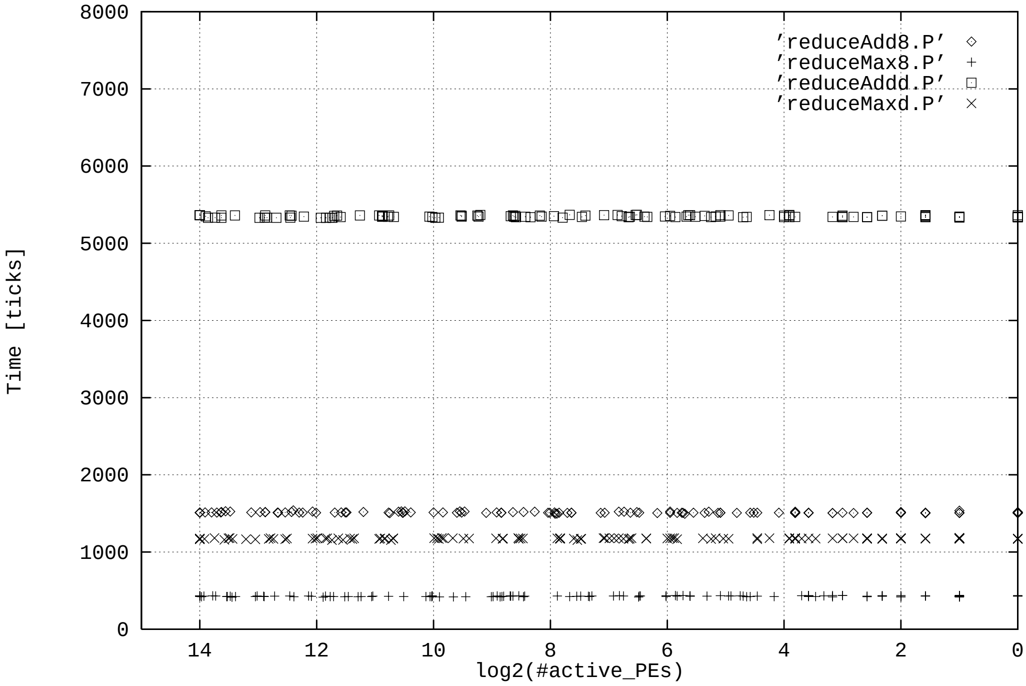


pp_xfetch and pp_xsend for packets of 4 byte and 32 byte, respectively. The seemin
pp_xfetch.32 at $x = 67$ is a not reproducible runaway which
the measurement progr

A comp
byte versus 32-byte packets (figure
show a difference by factor 4 for the distance-depende
constant effort for the larger packets. The constant time needed for a 32-byte pp_
7000 ticks. That means that it is faster to use the router (pp_rsend) for the same operation,
even for distance 1, if the number of participating PEs is less than 2^9 and the activation pattern
is random (or spread evenly). For distance 64 one can afford more than 2^{12} active PEs before
es slower!

ion library functions

untime behavior of various reduction library functions depending on PE
be dependent on the actual data used in the

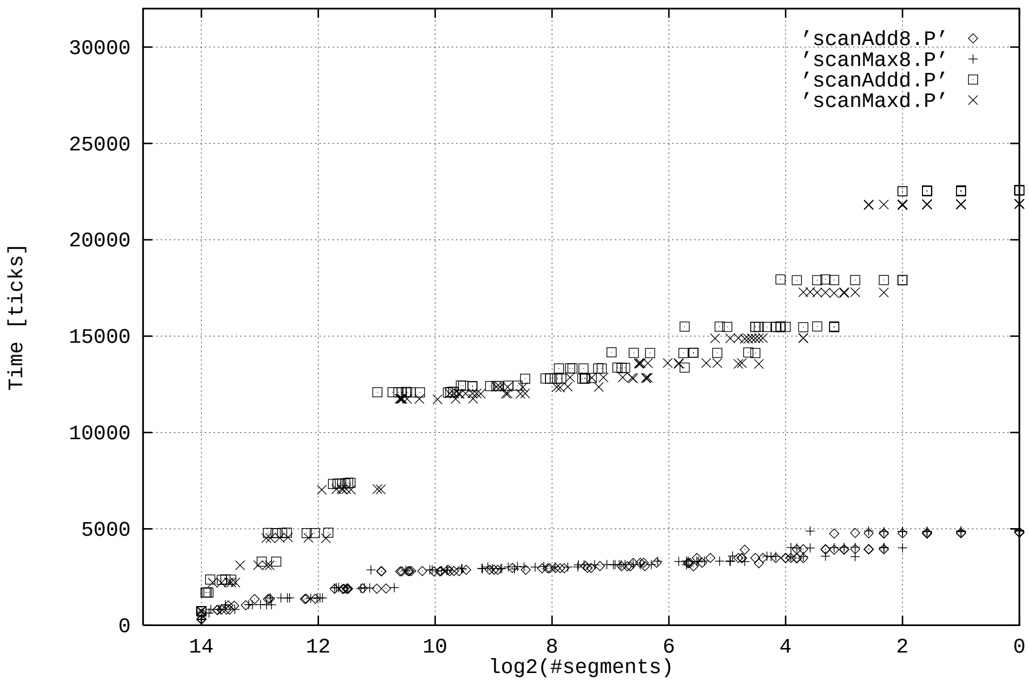


Runtime of the reduce library functions for adding `chars`, finding maximum `char`, ad
maximum `double` as a function of PE activity.

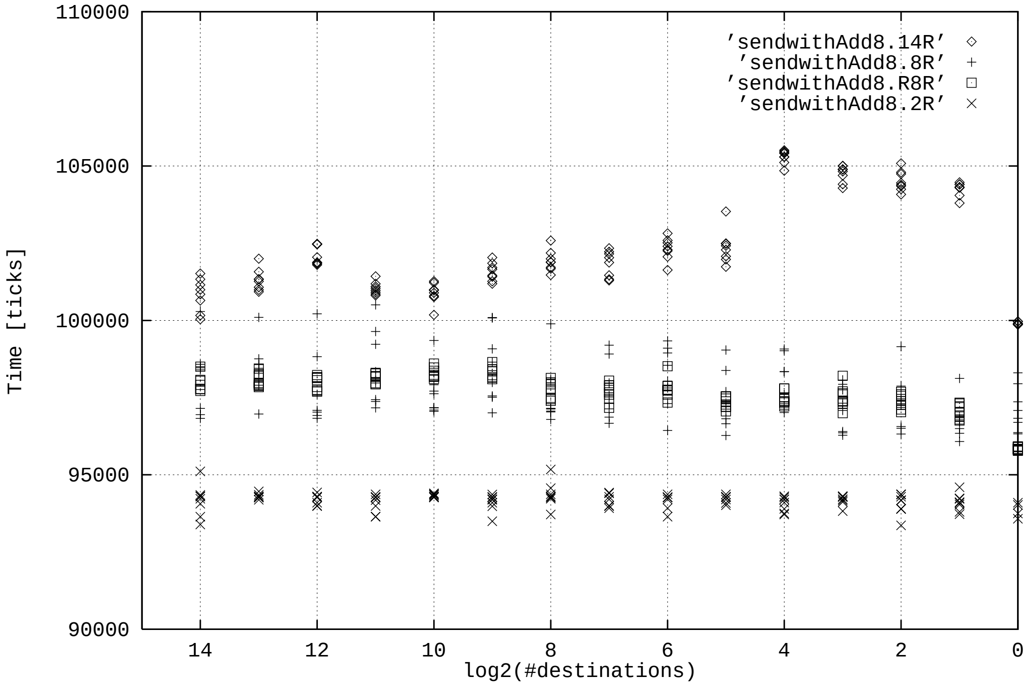
Figure 25: **reduc**

4.2 scan

The scan library functions, as the reduce functions, did not show any c
of PE activity. The dominant influence factor on the runtime of the scan functions is the size
of the segments that are scanned. In the experiment shown in figure 26, a varying number
of segment boundaries was thrown onto the PE array with even distribution and the resulting
I do not have an explanation for the quantum jumps in the curves, which could
t, but may stem from properties of the random
magnitude of time



Runtime of the scan library functions for adding chars, finding maximum char, adding do
maximum double as a function of the number of segments to be sc
randomly. Thus, the size of



Time for sendwith operations as a function of the number of destin
in a random pattern. The activat

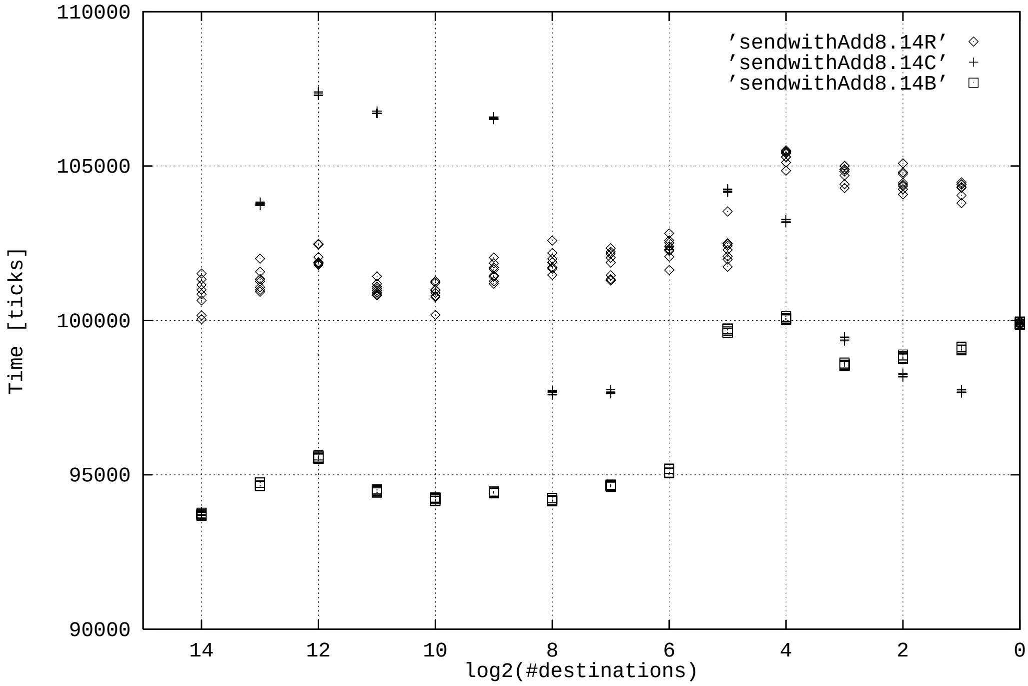
differ
small.

As a rule of thumb, a sendwith op
with all PEs participating.

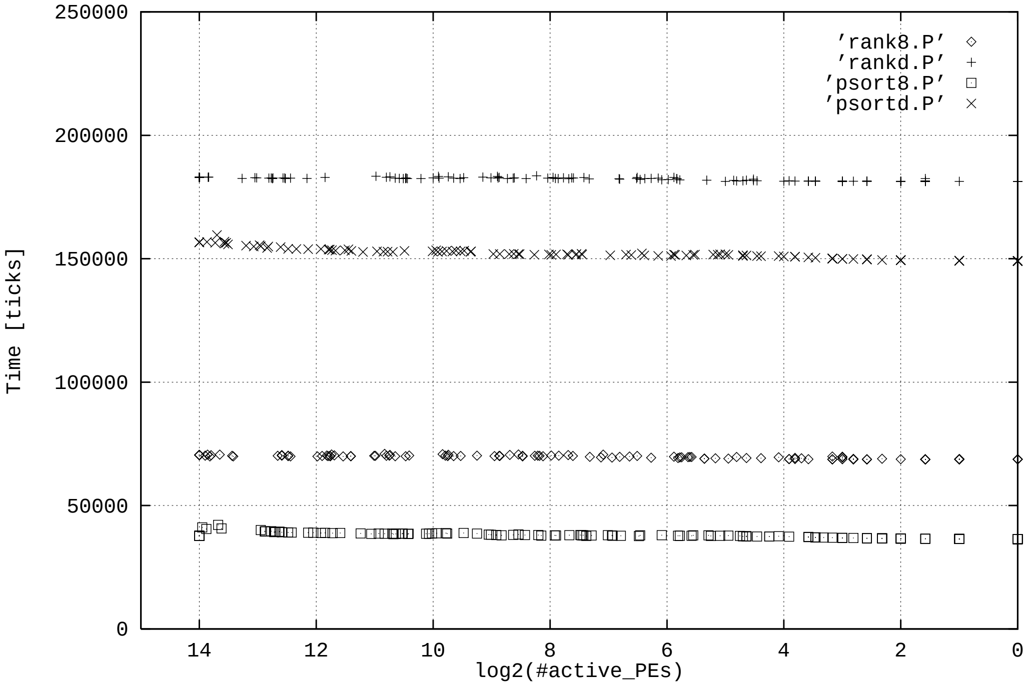
4.4 enumerate, selectOne, selectFirst

The diagram for the enumerate operation would show a straight l
than 7100 ticks. The case of all PEs being active is optimized and uses only 80 ticks. Si
diagram is completely boring, I left it out.

A similar statement is true for selectOne and selectFirst: both are independent of the activity
pattern and show little deviation. selectOne is used for performing a cast from plural to singular
all values are known to be identical), which was allowed to be written directly in the
ow. The operations can also be used to iterate sequentially
undefined order (selectOne). selec-
at there is



Time for sendwith operations as a function of the number of destination PEs. All PEs p
PEs are arranged in the following patterns: Even
distribute



The rank and psort routines applied to char and double, respectively.

Figure 30: **rank and psort**

5 Summary of Results

The measurements had some surprising results and some of them show significant differences in
a reaction to variations on innocent-looking parameters (especially the communica-

surements for actual programming decisions will in-

unication

A The timing program

To provide exact documentation of the experiments done, I simply print the whole timing
rogramI used. It consists of two modules: One of themcontains a small number of auxiliary
the actual timing code.

or better parallel randomnumber generation, random
off(for randomactivity

```
  */
  return (p_random() % 1000003);  /* use smallest prime greater 1e6 as modulus */
}
```

A.2 measure.h

```
/**************************************************************************
Project : MasPar communication timing program
Author  : Lutz Prechelt, Karlsruhe
Date    : 03.11.92
**************************************************************************/

/* not present in any header file although in V3.0 library: */
int             dpuTimerStart();
unsigned long   dpuTimerTicks();
double          dpuTimerElapsed();
int             selectFirst();
int             selectOne();

/* my own auxiliary routines: */
plural int      ca_every_nth (int n);
void            new_permutation (plural int *dest);
void            openfile (FILE **fp, char *filename, char *mode);
plural int      p_randm();
```

A.3 measure.m

```
/**************************************************************************
Project : MasPar test program for timings
Author  : Lutz Prechelt, Karlsruhe
Date    : 05.11.92
**************************************************************************/

#include <mpl.h>
#include <mp_libc.h>
#include <mp_libm.h>
#include <math.h>
#include <stdio.h>

#include "measure.h"


/* not present in any header file although in V3.0 library: */
int             dpuTimerStart();
unsigned long   dpuTimerTicks();
double          dpuTimerElapsed();
int             selectFirst();
int             selectOne();


visible extern int time (int*);
visible int     test_main ();
int             communication_tests ();

#define n 6

char              c1, c2;
short             s1, s2;
int               i1, i2;
float             f1, f2;
double            d1, d2;

plural char       pc1, pc2, pc[1024], pc_[1024], seg;
plural short      ps1, ps2, ps[n];
plural int        pi1, pi2, pi[n],              dest, dest2;
plural float      pf1, pf2, pf[n];
plural double     pd1, pd2, pd[n];

plural char       *pcp1, *pcp2;
plural short      *psp1, *psp2;
plural int        *pip1, *pip2;
plural float      *pfp1, *pfp2;
plural double     *pdp1, *pdp2;

plural char       *plural pcpp1, *plural pcpp2;
plural short      *plural pspp1, *plural pspp2;
plural int        *plural pipp1, *plural pipp2;
plural float      *plural pfpp1, *plural pfpp2;
plural double     *plural pdpp1, *plural pdpp2;

int time_nonempty;

/*-----------------------------------------------------------------------*/

visible int test_main ()
{
  /* this is the function that is called from the front-end program */
  printf ("StartACU:\n");
  srandom (callRequest (time, sizeof(int*), (int*)0));
```

```
  communication_tests();
  printf("EndACU ");
  return (0);
}

/*----------------------------------------------------------------------------*/

/* macros to perform a single measurement and write a protocol line */

#define measure_i(prg,lprob_act,size) \
  { int pecount = reduceMax32 (enumerate()) + 1; \
    __routerCount = 0; \
    dpuTimerStart (); \
    prg; \
    time_nonempty = dpuTimerTicks(); \
    fprintf (fp, "%5d %5.2f %2d %3d %6d %2d\n", \
             pecount, log10((double)pecount)/log10(2.0), \
             lprob_act, size, time_nonempty-60, __routerCount); \
  }

#define measure_f(prg,lprob_act,value) \
  { int pecount = reduceMax32 (enumerate()) + 1; \
    __routerCount = 0; \
    dpuTimerStart (); \
    prg; \
    time_nonempty = dpuTimerTicks(); \
    fprintf (fp, "%5d %5.2f %2d %5.2f %6d %2d\n", \
             pecount, log10((double)pecount)/log10(2.0), \
             lprob_act, value, time_nonempty-60, __routerCount); \
  }

/*----------------------------------------------------------------------------*/

int communication_tests ()
{
  /* Measure how long various communication operations take under various
     conditions.
     Each experiment is output into a different file.
     File naming conventions for router measurements: "OP.SZ.AC"
       OP is operation (send,fetch,ss_rsend, etc.)
       SZ is communicated data size per processor in bytes
       AC is Activity and Communication pattern
             A is R : regular pattern of activity (every nth active)
                  P : probabilistic
                  B : block (only first n active)
             C is P : random permutation of destinations
                  R : random destination
                  C : cycle: every nth
                  B : block: n neighbors
                  1 : all the same destination
                  S : shift (number gives length e.g. S100)
     Similar naming conventions are used for other operations.
  */
  FILE *fp = 0;
  register int i, j, k, l;
  dest = dest2 = iproc;
  new_permutation (&dest);  /* prepare permutation in dest variable */
  pc1 = pc2 = (plural char)dest;
  pc1 = router[dest].pc2;  /* initialize router ?*/
  pc1 = xnetE[1].pc1;      /* initialize xnet ? */
  pd1 = pd2 = (plural double)dest;
  p_srandom (dest);  /* init parallel random number generator */

  /********** determine permutation sensitivity **********/

  openfile (&fp, "send.4.AP1", "w");
  fprintf (fp, "# 32 bit send, all, different permutations (shifted powers):\n");
  for (k = 0; k < 100; k++) {
       dest = router[(dest+(plural)19) % nproc].dest;
       measure_i ((router[dest].pi1 = pi2), lnproc, k);
  }

  openfile (&fp, "send.4.AP2", "w");
  fprintf (fp, "# 32 bit send, all, different permutations (randomly new):\n");
  for (k = 0; k < 100; k++) {
       new_permutation (&dest);
       measure_i ((router[dest].pi1 = pi2), lnproc, k);
  }

  /********** xnet communication **********/

  /* no repetitions, since almost no deviations occur */

  openfile (&fp, "xnetEsend.1", "w");
  fprintf (fp, "# 8 bit xnet send:\n");
  for (j = 0; j <= nxproc; j++)
         measure_i ((xnetE[j].pc1 = pc2), j, 1);

  openfile (&fp, "xnetEsend.4", "w");
  fprintf (fp, "# 32 bit xnet send:\n");
  for (j = 0; j <= nxproc; j++)
         measure_i ((xnetE[j].pi1 = pi2), j, 4);

  openfile (&fp, "xnetEsend.8", "w");
  fprintf (fp, "# 64 bit xnet send:\n");
  for (j = 0; j <= nxproc; j++)
         measure_i ((xnetE[j].pd1 = pd2), j, 8);
```

```
openfile (&fp, "xnetEfetch.4", "w");
fprintf (fp, "# 32 bit xnet fetch:\n");
for (j = 0; j <= nxproc; j++)
       measure_i ((pi2 = xnetE[j].pi1), j, 4);

openfile (&fp, "ss_xfetch.4", "w");
fprintf (fp, "# 32 bit ss_xfetch:\n");
for (j = 0; j <= nxproc; j++)
       measure_i ((ss_xfetch (j, 0, pc, pc_, 4)), j, 4);

openfile (&fp, "ss_xfetch.4.Y20", "w");
fprintf (fp, "# 32 bit ss_xfetch:\n");
for (j = 0; j <= nxproc; j++)
       measure_i ((ss_xfetch (j, 20, pc, pc_, 4)), j, 4);

openfile (&fp, "ss_xsend.4", "w");
fprintf (fp, "# 32 bit ss_xsend:\n");
for (j = 0; j <= nxproc; j++)
       measure_i ((ss_xsend (j, 0, pc, pc_, 4)), j, 4);

openfile (&fp, "sp_xsend.4", "w");
fprintf (fp, "# 32 bit sp_xsend:\n");
for (j = 0; j <= nxproc; j++) {
       pcpp2 = pc + (p_randm() % (10*k+1));
       measure_i ((sp_xsend (j, 0, pc, pcpp2, 4)), j, 4);
}

openfile (&fp, "sp_xfetch.4", "w");
fprintf (fp, "# 32 bit sp_xfetch:\n");
for (j = 0; j <= nxproc; j++) {
       pcpp2 = pc + (p_randm() % (10*k+1));
       measure_i ((sp_xfetch (j, 0, pc, pcpp2, 4)), j, 4);
}

openfile (&fp, "ps_xsend.4", "w");
fprintf (fp, "# 32 bit ps_xsend:\n");
for (j = 0; j <= nxproc; j++) {
       pcpp1 = pc + (p_randm() % (10*k+1));
       measure_i ((ps_xsend (j, 0, pcpp1, pc, 4)), j, 4);
}

openfile (&fp, "ps_xfetch.4", "w");
fprintf (fp, "# 32 bit ps_xfetch:\n");
for (j = 0; j <= nxproc; j++) {
       pcpp1 = pc + (p_randm() % (10*k+1));
       measure_i ((ps_xfetch (j, 0, pcpp1, pc, 4)), j, 4);
}

openfile (&fp, "pp_xsend.4", "w");
fprintf (fp, "# 32 bit pp_xsend:\n");
for (j = 0; j <= nxproc; j++) {
       pcpp1 = pc + (p_randm() % (10*k+1));
       pcpp2 = pc + (p_randm() % (10*k+1));
       measure_i ((pp_xsend (j, 0, pcpp1, pcpp2, 4)), j, 4);
}

openfile (&fp, "pp_xfetch.4", "w");
fprintf (fp, "# 32 bit pp_xfetch:\n");
for (j = 0; j <= nxproc; j++) {
       pcpp1 = pc + (p_randm() % (10*k+1));
       pcpp2 = pc + (p_randm() % (10*k+1));
       measure_i ((pp_xfetch (j, 0, pcpp1, pcpp2, 4)), j, 4);
}

openfile (&fp, "pp_xfetch.4.Y20", "w");
fprintf (fp, "# 32 bit pp_xfetch:\n");
for (j = 0; j <= nxproc; j++) {
       pcpp1 = pc + (p_randm() % (10*k+1));
       pcpp2 = pc + (p_randm() % (10*k+1));
       measure_i ((pp_xfetch (j, 20, pcpp1, pcpp2, 4)), j, 4);
}

openfile (&fp, "pp_xsend.32", "w");
fprintf (fp, "# 32 byte pp_xsend:\n");
for (j = 0; j <= nxproc; j++) {
       pcpp1 = pc + (p_randm() % (10*k+1));
       pcpp2 = pc + (p_randm() % (10*k+1));
       measure_i ((pp_xsend (j, 0, pcpp1, pcpp2, 32)), j, 32);
}

openfile (&fp, "pp_xfetch.32", "w");
fprintf (fp, "# 32 byte pp_xfetch:\n");
for (j = 0; j <= nxproc; j++) {
       pcpp1 = pc + (p_randm() % (10*k+1));
       pcpp2 = pc + (p_randm() % (10*k+1));
       measure_i ((pp_xfetch (j, 0, pcpp1, pcpp2, 32)), j, 32);
}

/********** router with regular activity pattern **********/

openfile (&fp, "send.4.RP", "w");
fprintf (fp, "# 32 bit send, regular inactives:\n");
for (j = 0; j <= lnproc; j++) {
  for (k = 0; k < 8; k++) {
    dest = router[(dest+(plural)19) % nproc].dest;
    if (iproc % (1<<j) == 0)
      measure_i ((router[dest].pi1 = pi2), lnproc-j, 4);
  }
```

```
    }

    openfile (&fp, "send.4.BP", "w");
    fprintf (fp, "# 32 bit router send, first 2**n active:\n");
    for (j = 0; j <= lnproc; j++) {
      for (k = 0; k < 8; k++) {
         dest = router[(dest+(plural)19) % nproc].dest;
         if (iproc < (1<<j))
           measure_i ((router[dest].pi1 = pi2), lnproc-j, 4);
      }
      new_permutation (&dest);
    }

    openfile (&fp, "ss_rsend.32.RP", "w");
    fprintf (fp, "# 32 byte ss_rsend, regular inactives:\n");
    for (j = 0; j <= lnproc; j++) {
      for (k = 0; k < 8; k++) {
        dest = router[(dest+(plural)19) % nproc].dest;
        if (iproc % (1<<j) == 0)
           measure_i ((ss_rsend(dest, pc, pc_, 32)), lnproc-j, 32);
      }
    }

    openfile (&fp, "ss_rsend.32.BP", "w");
    fprintf (fp, "# 32 byte ss_rsend, first 2**n active:\n");
    for (j = 0; j <= lnproc; j++) {
      for (k = 0; k < 8; k++) {
         dest = router[(dest+(plural)19) % nproc].dest;
         if (iproc < (1<<j))
           measure_i ((ss_rsend(dest, pc, pc_, 32)), lnproc-j, 32);
      }
      new_permutation (&dest);
    }

    /********** router with random activity pattern **********/

    openfile (&fp, "send.1.PP", "w");
    fprintf (fp, "# 8 bit send, random inactives:\n");
    for (j = 0; j <= lnproc; j++) {
      for (k = 0; k < 8; k++) {
         dest = router[(dest+(plural)19) % nproc].dest;
         if (ca_every_nth (1<<j))
           measure_i ((router[dest].pc1 = pc2), lnproc-j, 1);
      }
    }

    openfile (&fp, "send.2.PP", "w");
    fprintf (fp, "# 16 bit send, random inactives:\n");
    for (j = 0; j <+ lnproc; j++) {
      for (k = 0; k < 8; k++) {
         dest = router[(dest+(plural)19) % nproc].dest;
         if (ca_every_nth (1<<j))
           measure_i ((router[dest].ps1 = ps2), lnproc-j, 2);
      }
    }

    openfile (&fp, "send.4.PP", "w");
    fprintf (fp, "# 32 bit send, random inactives:\n");
    for (j = 0; j <= lnproc; j++) {
      for (k = 0; k < 8; k++) {
         dest = router[(dest+(plural)19) % nproc].dest;
         if (ca_every_nth (1<<j))
           measure_i ((router[dest].pi1 = pi2), lnproc-j, 4);
      }
    }

    openfile (&fp, "send.8.PP", "w");
    fprintf (fp, "# 64 bit send, random inactives:\n");
    for (j = 0; j <= lnproc; j++) {
      for (k = 0; k < 8; k++) {
         dest = router[(dest+(plural)19) % nproc].dest;
         if (ca_every_nth (1<<j))
           measure_i ((router[dest].pd1 = pd2), lnproc-j, 8);
      }
    }

    openfile (&fp, "ss_rsend.4.PP", "w");
    fprintf (fp, "# 4 byte ss_rsend, random inactives:\n");
    for (j = 0; j <= lnproc; j++) {
      for (k = 0; k < 8; k++) {
         dest = router[(dest+(plural)19) % nproc].dest;
         if (ca_every_nth (1<<j))
           measure_i ((ss_rsend(dest, pc, pc_, 4)), lnproc-j, 4);
      }
    }

    openfile (&fp, "sp_rsend.4.PP", "w");
    fprintf (fp, "# 4 byte sp_rsend, random inactives:\n");
    for (j = 0; j <= lnproc; j++) {
      for (k = 0; k < 8; k++) {
         dest = router[(dest+(plural)19) % nproc].dest;
         pcpp2 = pc + (p_randm() % (10*k+1));
         if (ca_every_nth (1<<j))
           measure_i ((sp_rsend(dest, pc, pcpp2, 4)), lnproc-j, 4);
      }
    }

    openfile (&fp, "ps_rsend.4.PP", "w");
```

```
fprintf (fp, "# 4 byte ps_rsend, random inactives:\n");
for (j = 0; j <= lnproc; j++) {
  for (k = 0; k < 8; k++) {
     dest = router[(dest+(plural)19) % nproc].dest;
     pcpp1 = pc + (p_randm() % (10*k+1));
     if (ca_every_nth (1<<j))
       measure_i ((ps_rsend(dest, pcpp1, pc, 4)), lnproc-j, 4);
  }
}

openfile (&fp, "pp_rsend.4.PP", "w");
fprintf (fp, "# 4 byte pp_rsend, random inactives:\n");
for (j = 0; j <= lnproc; j++) {
  for (k = 0; k < 8; k++) {
     dest = router[(dest+(plural)19) % nproc].dest;
     pcpp1 = pc + (p_randm() % (10*k+1));
     pcpp2 = pc + (p_randm() % (10*k+1));
     if (ca_every_nth (1<<j))
       measure_i ((pp_rsend(dest, pcpp1, pcpp2, 4)), lnproc-j, 4);
  }
}

openfile (&fp, "ss_rfetch.4.PP", "w");
fprintf (fp, "# 4 byte ss_rfetch, random inactives:\n");
for (j = 0; j <= lnproc; j++) {
  for (k = 0; k < 8; k++) {
     dest = router[(dest+(plural)19) % nproc].dest;
     if (ca_every_nth (1<<j))
       measure_i ((ss_rfetch(dest, pc, pc_, 4)), lnproc-j, 4);
  }
}

openfile (&fp, "sp_rfetch.4.PP", "w");
fprintf (fp, "# 4 byte sp_rfetch, random inactives:\n");
for (j = 0; j <= lnproc; j++) {
  for (k = 0; k < 8; k++) {
     dest = router[(dest+(plural)19) % nproc].dest;
     pcpp2 = pc + (p_randm() % (10*k+1));
     if (ca_every_nth (1<<j))
       measure_i ((sp_rfetch(dest, pc, pcpp2, 4)), lnproc-j, 4);
  }
}

openfile (&fp, "ps_rfetch.4.PP", "w");
fprintf (fp, "# 4 byte ps_rfetch, random inactives:\n");
for (j = 0; j <= lnproc; j++) {
  for (k = 0; k < 8; k++) {
     dest = router[(dest+(plural)19) % nproc].dest;
     pcpp1 = pc + (p_randm() % (10*k+1));
     if (ca_every_nth (1<<j))
       measure_i ((ps_rfetch(dest, pcpp1, pc, 4)), lnproc-j, 4);
  }
}

openfile (&fp, "pp_rfetch.4.PP", "w");
fprintf (fp, "# 4 byte pp_rfetch, random inactives:\n");
for (j = 0; j <= lnproc; j++) {
  for (k = 0; k < 8; k++) {
     dest = router[(dest+(plural)19) % nproc].dest;
     pcpp1 = pc + (p_randm() % (10*k+1));
     pcpp2 = pc + (p_randm() % (10*k+1));
     if (ca_every_nth (1<<j))
       measure_i ((pp_rfetch(dest, pcpp1, pcpp2, 4)), lnproc-j, 4);
  }
}

openfile (&fp, "ss_rsend.32.PP", "w");
fprintf (fp, "# 32 byte ss_rsend, random inactives:\n");
for (j = 0; j <= lnproc; j++) {
  for (k = 0; k < 8; k++) {
     dest = router[(dest+(plural)19) % nproc].dest;
     if (ca_every_nth (1<<j))
       measure_i ((ss_rsend(dest, pc, pc_, 32)), lnproc-j, 32);
  }
}

openfile (&fp, "sp_rsend.32.PP", "w");
fprintf (fp, "# 32 byte sp_rsend, random inactives:\n");
for (j = 0; j <= lnproc; j++) {
  for (k = 0; k < 8; k++) {
     dest = router[(dest+(plural)19) % nproc].dest;
     pcpp2 = pc + (p_randm() % (10*k+1));
     if (ca_every_nth (1<<j))
       measure_i ((sp_rsend(dest, pc, pcpp2, 32)), lnproc-j, 32);
  }
}

openfile (&fp, "ps_rsend.32.PP", "w");
fprintf (fp, "# 32 byte ps_rsend, random inactives:\n");
for (j = 0; j <= lnproc; j++) {
  for (k = 0; k < 8; k++) {
     dest = router[(dest+(plural)19) % nproc].dest;
     pcpp1 = pc + (p_randm() % (10*k+1));
     if (ca_every_nth (1<<j))
       measure_i ((ps_rsend(dest, pcpp1, pc, 32)), lnproc-j, 32);
  }
}
```

```
openfile (&fp, "pp_rsend.32.PP", "w");
fprintf (fp, "# 32 byte pp_rsend, random inactives:\n");
for (j = 0; j <= lnproc; j++) {
  for (k = 0; k < 8; k++) {
     dest = router[(dest+(plural)19) % nproc].dest;
     pcpp1 = pc + (p_randm() % (10*k+1));
     pcpp2 = pc + (p_randm() % (10*k+1));
     if (ca_every_nth (1<<j))
       measure_i ((pp_rsend(dest, pcpp1, pcpp2, 32)), lnproc-j, 32);
  }
}

openfile (&fp, "ss_rsend.128.PP", "w");
fprintf (fp, "# 128 byte ss_rsend, random inactives:\n");
for (j = 0; j <= lnproc; j++) {
  for (k = 0; k < 8; k++) {
     dest = router[(dest+(plural)19) % nproc].dest;
     if (ca_every_nth (1<<j))
       measure_i ((ss_rsend(dest, pc, pc_, 128)), lnproc-j, 128);
  }
}

openfile (&fp, "ss_rsend.256.PP", "w");
fprintf (fp, "# 256 byte ss_rsend, random inactives:\n");
for (j = 0; j <= lnproc; j++) {
  for (k = 0; k < 8; k++) {
     dest = router[(dest+(plural)19) % nproc].dest;
     if (ca_every_nth (1<<j))
       measure_i ((ss_rsend(dest, pc, pc_, 256)), lnproc-j, 256);
  }
}

openfile (&fp, "fetch.1.PP", "w");
fprintf (fp, "# 8 bit fetch, random inactives:\n");
for (j = 0; j <= lnproc; j++) {
  for (k = 0; k < 8; k++) {
     dest = router[(dest+(plural)19) % nproc].dest;
     if (ca_every_nth (1<<j))
       measure_i ((pc2 = router[dest].pc1), lnproc-j, 1);
  }
}

openfile (&fp, "fetch.2.PP", "w");
fprintf (fp, "# 16 bit fetch, random inactives:\n");
for (j = 0; j <= lnproc; j++) {
  for (k = 0; k < 8; k++) {
     dest = router[(dest+(plural)19) % nproc].dest;
     if (ca_every_nth (1<<j))
       measure_i ((ps2 = router[dest].ps1), lnproc-j, 2);
  }
}

openfile (&fp, "fetch.4.PP", "w");
fprintf (fp, "# 32 bit fetch, random inactives:\n");
for (j = 0; j <= lnproc; j++) {
  for (k = 0; k < 8; k++) {
     dest = router[(dest+(plural)19) % nproc].dest;
     if (ca_every_nth (1<<j))
       measure_i ((pi2 = router[dest].pi1), lnproc-j, 4);
  }
}

openfile (&fp, "fetch.4.P1", "w");
fprintf (fp, "# 32 bit fetch from PE 1, random inactives:\n");
dest2 = (plural)1;
for (j = 0; j <= lnproc; j++) {
  for (k = 0; k < 8; k++) {
     if (ca_every_nth (1<<j))
       measure_i ((pi2 = router[dest2].pi1), lnproc-j, 4);
  }
}


openfile (&fp, "fetch.4.PS1", "w");
fprintf (fp, "# 32 bit fetch shift 1, random inactives:\n");
for (j = 0; j <= lnproc; j++) {
  for (k = 0; k < 8; k++) {
     dest2 = (iproc + 100) % nproc;
     if (ca_every_nth (1<<j))
       measure_i ((pi2 = router[dest2].pi1), lnproc-j, 4);
  }
}

openfile (&fp, "fetch.4.PS100", "w");
fprintf (fp, "# 32 bit fetch shift 100, random inactives:\n");
for (j = 0; j <= lnproc; j++) {
  for (k = 0; k < 8; k++) {
     dest2 = (iproc + 100) % nproc;
     if (ca_every_nth (1<<j))
       measure_i ((pi2 = router[dest2].pi1), lnproc-j, 4);
  }
}

openfile (&fp, "fetch.4.PR", "w");
fprintf (fp, "# 32 bit fetch from random PEs, random inactives:\n");
for (j = 0; j <= lnproc; j++) {
  for (k = 0; k < 8; k++) {
     dest2 = p_randm() % nproc;
```

```
      if (ca_every_nth (1<<j))
        measure_i ((pi2 = router[dest2].pi1), lnproc-j, 4);
  }
}

openfile (&fp, "fetch.8.PP", "w");
fprintf (fp, "# 64 bit fetch, random inactives:\n");
for (j = 0; j <= lnproc; j++) {
  for (k = 0; k < 8; k++) {
      dest = router[(dest+(plural)19) % nproc].dest;
      if (ca_every_nth (1<<j))
        measure_i ((pd2 = router[dest].pd1), lnproc-j, 8);
  }
}

/********** enumerate() **********/

openfile (&fp, "enumerate.P", "w");
fprintf (fp, "# enumerate(), random inactives:\n");
for (j = 0; j <= lnproc; j++) {
  for (k = 0; k < 8; k++) {
      if (ca_every_nth (1<<j))
        measure_i ((pi1 = enumerate()), lnproc-j, 2);
  }
}

/********** selectFirst(), selectOne() **********/

openfile (&fp, "selectFirst.P", "w");
fprintf (fp, "# selectFirst(), random inactives:\n");
for (j = 0; j <= lnproc; j++) {
  for (k = 0; k < 8; k++) {
      if (ca_every_nth (1<<j))
        measure_i ((i = selectFirst()), lnproc-j, 2);
  }
}

openfile (&fp, "selectOne.P", "w");
fprintf (fp, "# selectOne(), random inactives:\n");
for (j = 0; j <= lnproc; j++) {
  for (k = 0; k < 8; k++) {
      if (ca_every_nth (1<<j))
        measure_i ((i = selectOne()), lnproc-j, 2);
  }
}

/********** psort(), rank() **********/

openfile (&fp, "psort8.P", "w");
fprintf (fp, "# psort8(), random inactives:\n");
for (j = 0; j <= lnproc; j++) {
  for (k = 0; k < 8; k++) {
      if (ca_every_nth (1<<j))
        measure_i ((pc1 = psort8(pc2)), lnproc-j, 1);
  }
}

openfile (&fp, "psortd.P", "w");
fprintf (fp, "# psortd(), random inactives:\n");
for (j = 0; j <= lnproc; j++) {
  for (k = 0; k < 8; k++) {
      pd2 = (plural double)(p_random() % (plural)1000);
      if (ca_every_nth (1<<j))
        measure_i ((pd1 = psortd(pd2)), lnproc-j, 8);
  }
}

openfile (&fp, "rank8.P", "w");
fprintf (fp, "# rank8(), random inactives:\n");
for (j = 0; j <= lnproc; j++) {
  for (k = 0; k < 8; k++) {
      if (ca_every_nth (1<<j))
        measure_i ((pi1 = rank8(pc1)), lnproc-j, 2);
  }
}

openfile (&fp, "rankd.P", "w");
fprintf (fp, "# rankd(), random inactives:\n");
for (j = 0; j <= lnproc; j++) {
  for (k = 0; k < 8; k++) {
      pd2 = (plural double)(p_random() % (plural)1000);
      if (ca_every_nth (1<<j))
        measure_i ((pi1 = rankd(pd1)), lnproc-j, 2);
  }
}

/********** reduce() **********/

openfile (&fp, "reduceAdd8.P", "w");
fprintf (fp, "# reduceAdd8(), random inactives:\n");
for (j = 0; j <= lnproc; j++) {
  for (k = 0; k < 8; k++) {
      if (ca_every_nth (1<<j))
        measure_i ((c1 = reduceAdd8(pc2)), lnproc-j, 1);
  }
}

openfile (&fp, "reduceMax8.P", "w");
```

```
fprintf (fp, "# reduceMax8(), random inactives:\n");
for (j = 0; j <= lnproc; j++) {
  for (k = 0; k < 8; k++) {
     if (ca_every_nth (1<<j))
       measure_i ((c1 = reduceMax8(pc2)), lnproc-j, 1);
  }
}

openfile (&fp, "reduceAddd.P", "w");
fprintf (fp, "# reduceAddd(), random inactives:\n");
for (j = 0; j <= lnproc; j++) {
  for (k = 0; k < 8; k++) {
     pd2 = (plural double)(p_random() % (plural)1000);
     if (ca_every_nth (1<<j))
       measure_i ((d1 = reduceAddd(pd2)), lnproc-j, 8);
  }
}

openfile (&fp, "reduceMaxd.P", "w");
fprintf (fp, "# reduceMaxd(), random inactives:\n");
for (j = 0; j <= lnproc; j++) {
  for (k = 0; k < 8; k++) {
     pd2 = (plural double)p_random();
     if (ca_every_nth (1<<j))
       measure_i ((d1 = reduceMaxd(pd2)), lnproc-j, 8);
  }
}

/********** scan() **********/

openfile (&fp, "scanAdd8.P", "w");
fprintf (fp, "# scanAdd8(), random inactives:\n");
for (j = 0; j <= lnproc; j++) {
  for (k = 0; k < 8; k++) {
     seg =  ca_every_nth (1<<j);
     measure_f ((pc1 = scanAdd8(pc2, seg)), lnproc,
             log((float)(1+reduceAdd16((plural short)seg)))/log(2.0));
  }
}

openfile (&fp, "scanAddd.P", "w");
fprintf (fp, "# scanAddd(), random inactives:\n");
for (j = 0; j <= lnproc; j++) {
  for (k = 0; k < 8; k++) {
     seg =  ca_every_nth (1<<j);
     measure_f ((pd1 = scanAddd(pd2, seg)), lnproc,
             log((float)(1+reduceAdd16((plural short)seg)))/log(2.0));
  }
}

openfile (&fp, "scanMax8.P", "w");
fprintf (fp, "# scanMax8(), random inactives:\n");
for (j = 0; j <= lnproc; j++) {
  for (k = 0; k < 8; k++) {
     seg =  ca_every_nth (1<<j);
     measure_f ((pc1 = scanMax8(pc2, seg)), lnproc,
             log((float)(1+reduceAdd16((plural short)seg)))/log(2.0));
  }
}

openfile (&fp, "scanMaxd.P", "w");
fprintf (fp, "# scanMaxd(), random inactives:\n");
for (j = 0; j <= lnproc; j++) {
  for (k = 0; k < 8; k++) {
     pd2 = (plural double)(p_random() % (plural)1000);
     seg =  ca_every_nth (1<<j);
     measure_f ((pd1 = scanMaxd(pd2, seg)), lnproc,
             log((float)(1+reduceAdd16((plural short)seg)))/log(2.0));
  }
}

/********** sendwith() **********/

openfile (&fp, "sendwithAdd8.14R", "w");
fprintf (fp, "# sendwithAdd8(), all, random destinations:\n");
for (j = 0; j <= lnproc; j++) {
  for (k = 0; k < 8; k++) {
     dest2 =  p_randm() % (plural)(1<<j);
     measure_i ((pc1 = sendwithAdd8(pc2, dest2)), lnproc, j);
  }
}

openfile (&fp, "sendwithAdd8.14C", "w");
fprintf (fp, "# sendwithAdd8(), all, regular destinations cycle:\n");
for (j = 0; j <= lnproc; j++) {
  for (k = 0; k < 8; k++) {
     dest2 =  iproc % (plural)(1<<j);
     measure_i ((pc1 = sendwithAdd8(pc2, dest2)), lnproc, j);
  }
}

openfile (&fp, "sendwithAdd8.14B", "w");
fprintf (fp, "# sendwithAdd8(), all, regular destinations block:\n");
for (j = 0; j <= lnproc; j++) {
  for (k = 0; k < 8; k++) {
     dest2 =  iproc >> (plural)(lnproc-j);
     measure_i ((pc1 = sendwithAdd8(pc2, dest2)), lnproc, j);
  }
```

```
  }

  openfile (&fp, "sendwithAdd8.8R", "w");
  fprintf (fp, "# sendwithAdd8(), 256 random active, random destinations:\n");
  for (j = 0; j <= lnproc; j++) {
    for (k = 0; k < 8; k++) {
       dest2 =  p_randm() % (plural)(1<<j);
       if (ca_every_nth (1<<6))
         measure_i ((pc1 = sendwithAdd8(pc2, dest2)), 8, j);
    }
  }

  openfile (&fp, "sendwithAdd8.R8R", "w");
  fprintf (fp, "# sendwithAdd8(), 256 regular active, random destinations:\n");
  for (j = 0; j <= lnproc; j++) {
    for (k = 0; k < 8; k++) {
       dest2 =  p_randm() % (plural)(1<<j);
       if (iproc % (1<<6) == 0)
         measure_i ((pc1 = sendwithAdd8(pc2, dest2)), 8, j);
    }
  }

  openfile (&fp, "sendwithAdd8.2R", "w");
  fprintf (fp, "# sendwithAdd8(), 4 random active, random destinations:\n");
  for (j = 0; j <= lnproc; j++) {
    for (k = 0; k < 8; k++) {
       dest2 =  p_randm() % (plural)(1<<j);
       if (ca_every_nth (1<<12))
         measure_i ((pc1 = sendwithAdd8(pc2, dest2)), 2, j);
    }
  }

  openfile (&fp, "sendwithMax8.14R", "w");
  fprintf (fp, "# sendwithMax8(), all, random destinations:\n");
  for (j = 0; j <= lnproc; j++) {
    for (k = 0; k < 8; k++) {
       dest2 =  p_randm() % (plural)(1<<j);
       measure_i ((pc1 = sendwithMax8(pc2, dest2)), lnproc, j);
    }
  }

  openfile (&fp, "sendwithAddd.14R", "w");
  fprintf (fp, "# sendwithAddd(), all, random destinations:\n");
  for (j = 0; j <= lnproc; j++) {
    for (k = 0; k < 8; k++) {
       pd2 = (plural double)(p_random() % (plural)1000);
       dest2 =  p_randm() % (plural)(1<<j);
       measure_i ((pd1 = sendwithAddd(pd2, dest2)), lnproc, j);
    }
  }

  openfile (&fp, "sendwithMaxd.14R", "w");
  fprintf (fp, "# sendwithMaxd(), all, random destinations:\n");
  for (j = 0; j <= lnproc; j++) {
    for (k = 0; k < 8; k++) {
       pd2 = (plural double)(p_random() % (plural)1000);
       dest2 =  p_randm() % (plural)(1<<j);
       measure_i ((pd1 = sendwithMaxd(pd2, dest2)), lnproc, j);
    }
  }

  return (0);
}
```