McGill University

Communication Optimizations for Parallel C Programs

Yingchun Zhu and Laurie Hendren

McGill University

Outline

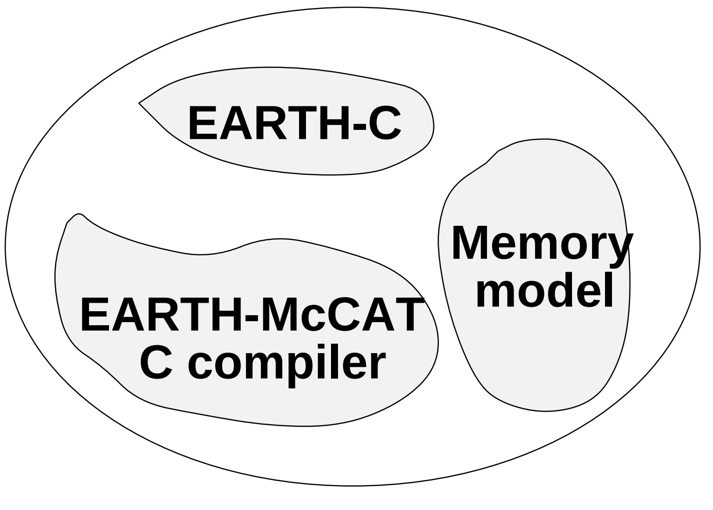
1. Introduction
2. Motivating Examples
3. Communication Optimizations
4. Experimental Results
5. Related Work
6. Conclusions and Future Work

Introduction I: EARTH-McCAT Project Overview

Project Goal

- **Compile** *parallel C programs* **for** *distributed memory architectures*
- **Focus on** *irregular programs* **using** *dynamic data structures*

Programming Model



Introduction II: EARTH-C: a parallel dialect to C

Parallelism:

```
{^  foo( t->left) ;
    foo(t->right);
 ^}
```

Computation distribution:

```
boo();
bar() @ home ;
baz(x) @ ownerof (x);
```

Local / Remote memory accesses:

struct node *local* * p;

EARTH-C & Threaded-C examples

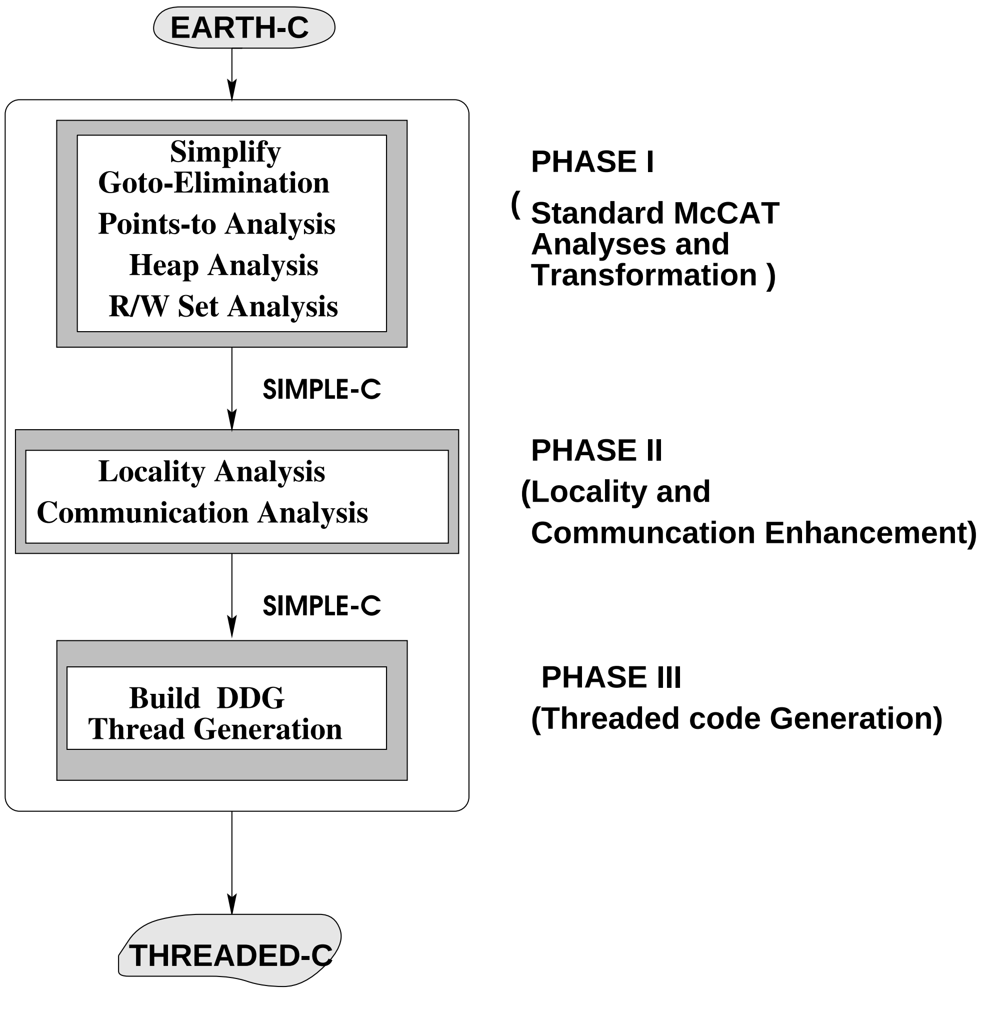
```
int TreeAdd( tree *t)
{
  int lval, rval, val, result;
   if (t != NULL)
        result = 0;
   else
    {
       {^ lval = TreeAdd(t->left);
           rval = TreeAdd(t->right);
        ^}
         val = t->value;
        result = lval + rval +val;
     }
   return(result);
  }
```

(a) EARTH-C program

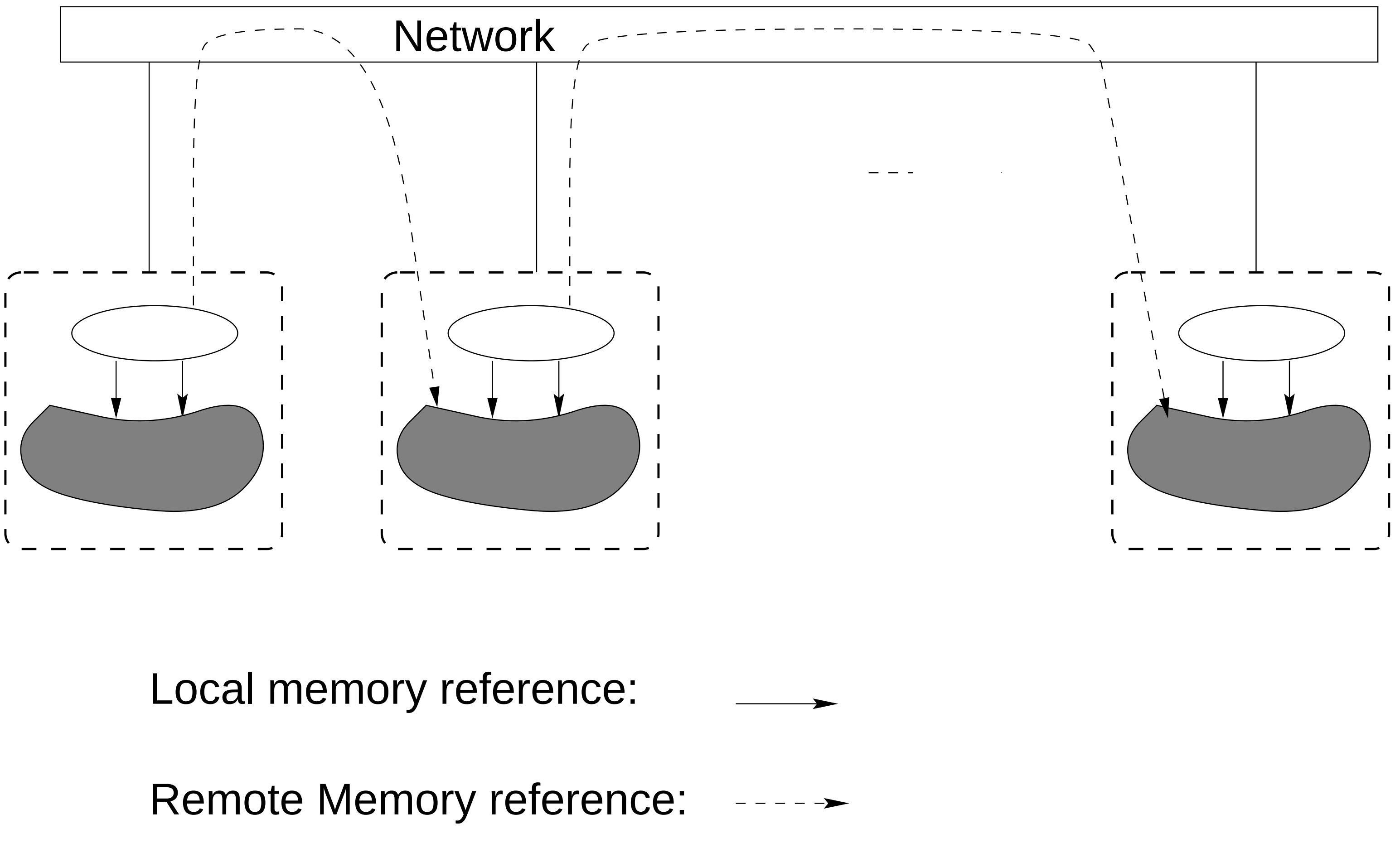
```
Threaded TreeAdd(int *result,  SLOT *done, tree *t)
{
    INIT_SYNC (0, 2, 2, 1);
    INIT_SYNC(1, 3, 3, 2 );
    if ( t == NULL)
       DATA_RSYNC(0, result, done);
    else {
       GET_SYNC(&(t->left), tmp1, 0);
       GET_SYNC(&(t->right), tmp2, 0);
       GET_SYNC(&(t->val), val, 1);
       END_THREAD();
     THREAD_1:
       TOKEN(TreeAdd, &lval, SLOT_ADR(1), tmp1);
       TOKEN(TreeAdd, &rval, SLOT_ADR(1), tmp2);
      END_THREAD();
     THREAD_2:
       DATA_RSYNC(lval+rval+val, result, done);
     }
    END_FUNCTION();
  }
```

(b) Threaded-C program

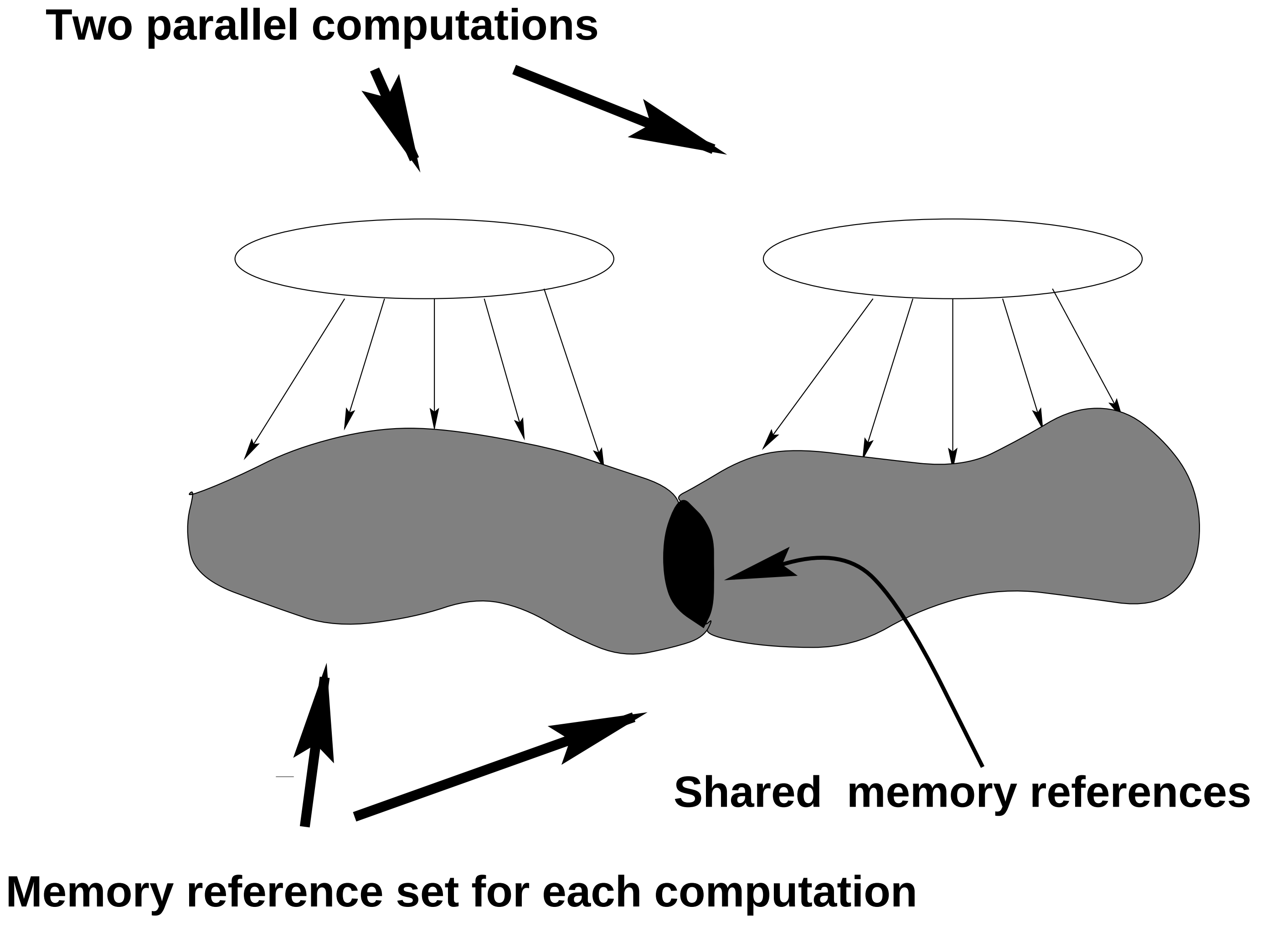
Introduction III: EARTH-McCAT C compiler



Memory Model I: Local vs. Remote



Memory Model II : Interference Rules



Example 1: Optimizing remote reads

```
dist_p = sqrt (
p->x * p->x
+ p->y * p->y );
```

Simplifying →

```
temp1 = p->x ;
temp2 = p->x ;
temp3 = temp1 * temp2 ;
temp4 = p->y ;
temp5 = p->y ;
temp6 = temp4 * temp5 ;
temp7 = temp3 + temp6 ;
dist_p = sqrt ( temp_7) ;
```

Example 1: Optimizing remote reads

```
temp1 = p->x ;
temp2 = p->x ;
temp3 = temp1 * temp2 ;
temp4 = p->y ;
temp5 = p->y ;
temp6 = temp4 * temp5 ;
temp7 = temp3 + temp6 ;
dist_p = sqrt ( temp_7) ;
```

Collecting and Moving Reads

```
comm1 = p->x ;
comm2 = p->y ;
temp3 = comm1 * comm1;
temp6 = comm2 * comm2 ;
temp7 = temp3 + temp6 ;
dist_p = sqrt ( temp_7) ;
```

Example 1: Optimizing remote reads

```
comm1 = p->x ;
comm2 = p->y ;
temp3 = comm1 * comm1;
temp6 = comm2 * comm2 ;
temp7 = temp3 + temp6 ;
dist_p = sqrt ( temp_7) ;
```

Blocking → **Reads**

```
blkmov(p, &comm, sizeof(Point)) ;
temp3 = comm.x * comm.x ;
temp6 = comm.y * comm.y ;
temp7 = temp3 + temp6 ;
dist_p = sqrt ( temp_7) ;
```

Example 2: Optimizing Remote Reads/Writes

```
p->x = scale ( p->x , k) ;
p->y = scale ( p->y , k) ;
```

Simplifying

```
temp1 = p->x ;
temp2 = scale ( temp1, k) ;
p->x = temp2 ;
temp3 = p->y ;
temp4 = scale ( temp3, k) ;
p->y = temp4 ;
```

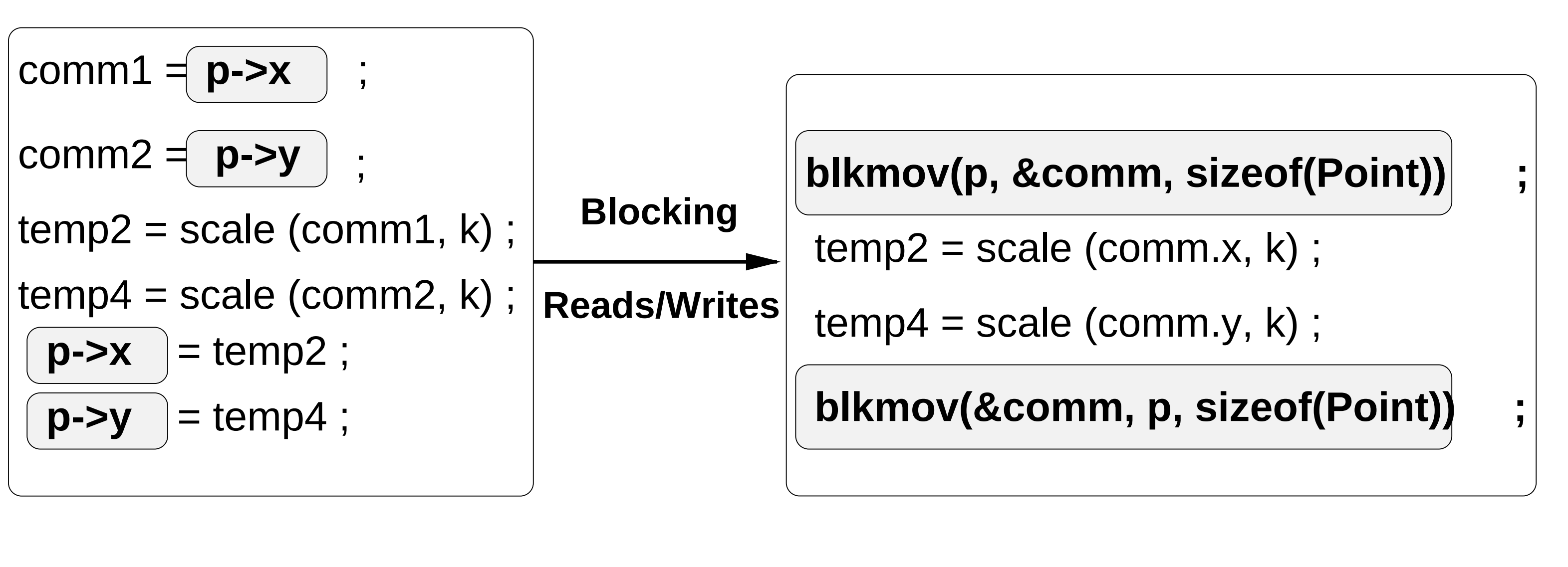
Example 2: Optimizing Remote Reads/Writes

```
temp1 = p->x ;
temp2 = scale ( temp1, k) ;
p->x = temp2 ;
temp3 = p->y ;
temp4 = scale ( temp3, k) ;
p->y = temp4 ;
```

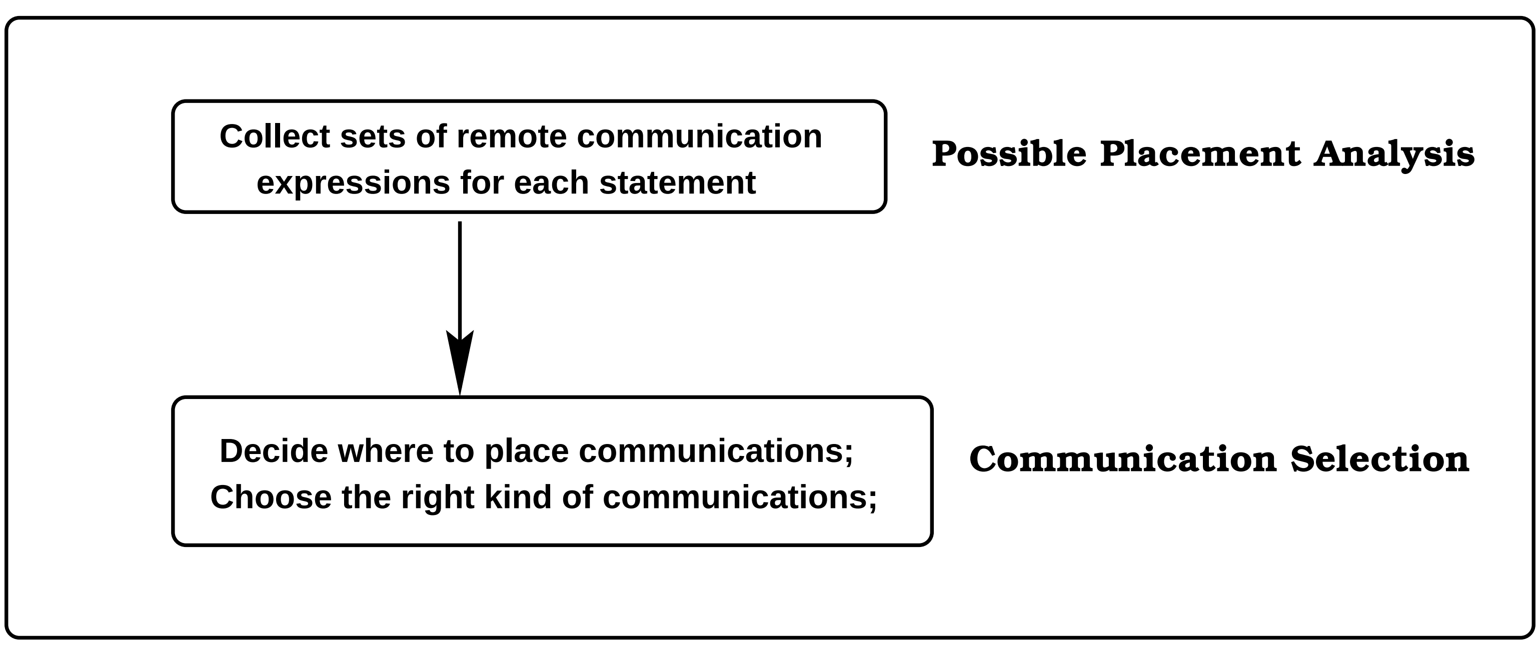
Moving Reads/Writes

```
comm1 = p->x ;
comm2 = p->y ;
temp2 = scale (comm1, k) ;
temp4 = scale (comm2, k) ;
p->x = temp2 ;
p->y = temp4 ;
```

Example 2: Optimizing Remote Reads/Writes



Communication Optimization Algorithm



Possible placement analysis

Collect **remote read/write set** for each statement

Set of **remote communication expressions**

A 4-tuple (p, f, n, Dlist) where:

- p: a pointer variable;
- f: the field of a struct;
- n: an estimated frequency;
- Dlist: A set of statement labels;

Example:

S1:	x = p->a;	Remote Read Set { (p->a, 1, S1) }
S2:	p->b = y;	Remote Write Set { (p->b, 1, S2) }

Propagating reads for sequence statements

{ ***(list->head, 1, S1)*** }

S1: l = list->head ;

{ *(l->x, 1, S6), (l->q, 1, S4),* ***(l->p, 1, S2)*** }

S2: tmp1 = l->p;

{ *(l->x, 1, S6), (l->q, 1, S4)* }

S3: tmp2 = tmp1 * delta;

{ *(l->x, 1, S6),* ***(l->q, 1, S4)*** }

S4: tmp3 = l->q;

{ *(l->x, 1, S6)* }

S5: l->q = tmp3 + tmp2;

{ ***(l->x, 1, S6)*** }

S6: tmp4 = l->x;

{}

S7: l->x = r * tmp4;

When is it safe to propagate reads?

Conditions:

1. **Base variable is not written**
2. **Field reference is not written through alias**

{}

S1: p = head;

{ (p->y, 1, S4) }

S2: t->x = 10;

{ (p->y, 1, S4), (p->x, 1, S3) }

S3: dx = p->x;

S4: dy = p->y;

Assuming t and p are aliased

When is it safe to propagate writes?

Conditions:

1. **Base variable is not written**
2. **Field reference is not read/written through alias**

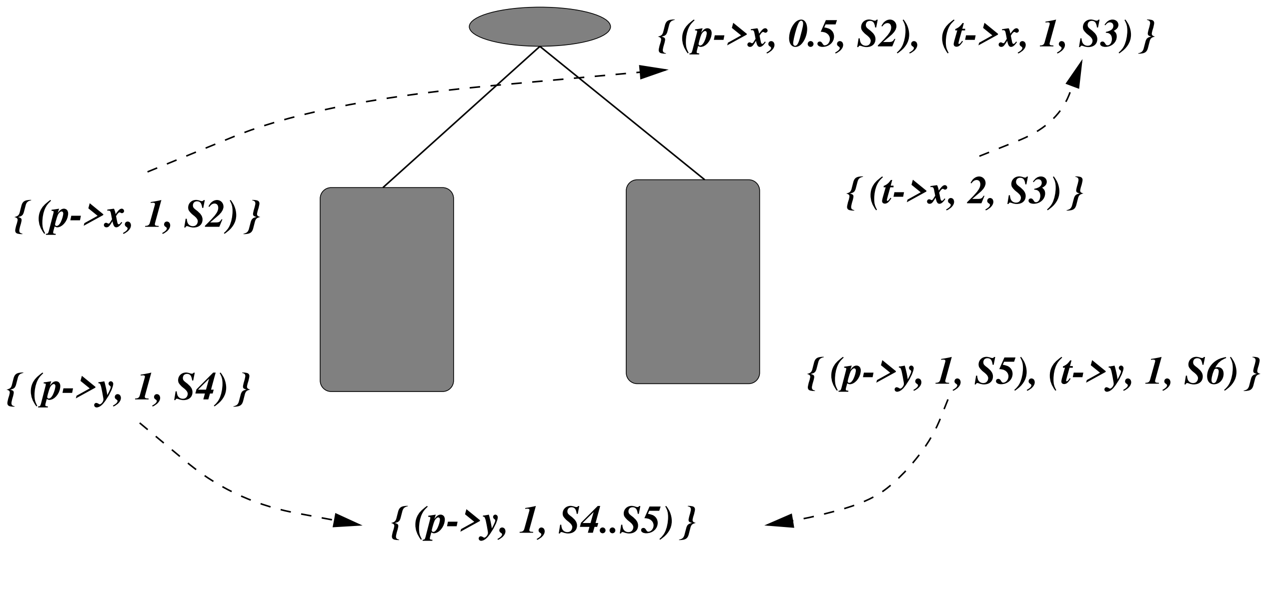
```
S1:  p->x = dx ;
                 { (p->x, 1, S1) }
S2:  tx = t->x ;
                 {}
S3:  p->y = dy ;
                 { (p->y, 1, S3) }
S4:  p = p->next ;
                 { }
```

Assuming t and p are aliased

Propagating reads for loops

```
                                    { (t->offset,  10 ,S2) }
S1:  while  ( p!= NULL )
        {   { (p->next, 1, S5),  (p->y, 1, S4),  (p->x, 1, S3),   (t->offset, 1, S2) }
S2:        s = t->offset;
                                 { (p->next, 1, S5),  (p->y, 1, S4),  (p->x, 1, S3)  }
S3:        dx = p->x;
                                 { (p->next, 1, S5),  (p->y, 1, S4) }
S4:        dy = p->y;
                                 {  (p->next, 1, S5) }
S5:        p = p->next;
        }
```

Propagating reads for branches



Communication Selection

- Traverse the program top-down to select remote reads, bottom-up to select remote writes;
- At each statement, check remote read/write set:
  - Remove any remotes already selected;
  - For the remaining remotes, decide:
    - Whether to place remotes at the spot using frequency information;
    - Whether to blkmov or pipeline them based on architecture information;

Select Reads

{ (t->x, 0.5, S5), ***(list->head, 1, S1)*** *}*

S1: l = list->head ;

{(t->x, 0.5, S5), ***(l->x, 0.5, S6), (l->q, l, S3) , (l->p,1, S2) }***

S2: tmp1 = l->p;

{(t->x, 0.5, S5), (l->x, 0.5, S6), (l->q, 1, S3) }

S3: tmp3 = l->q;

{(t->x, 0.5, S5), (l->x, 0.5, S6) }

S4: if (t != NULL)

{(t->x, 1, S5) }

S5: s = t->x;

else

{ (l->x, 1, S6) }

S6: s = l->x;

{}

S7: l->q = tmp3 + s;

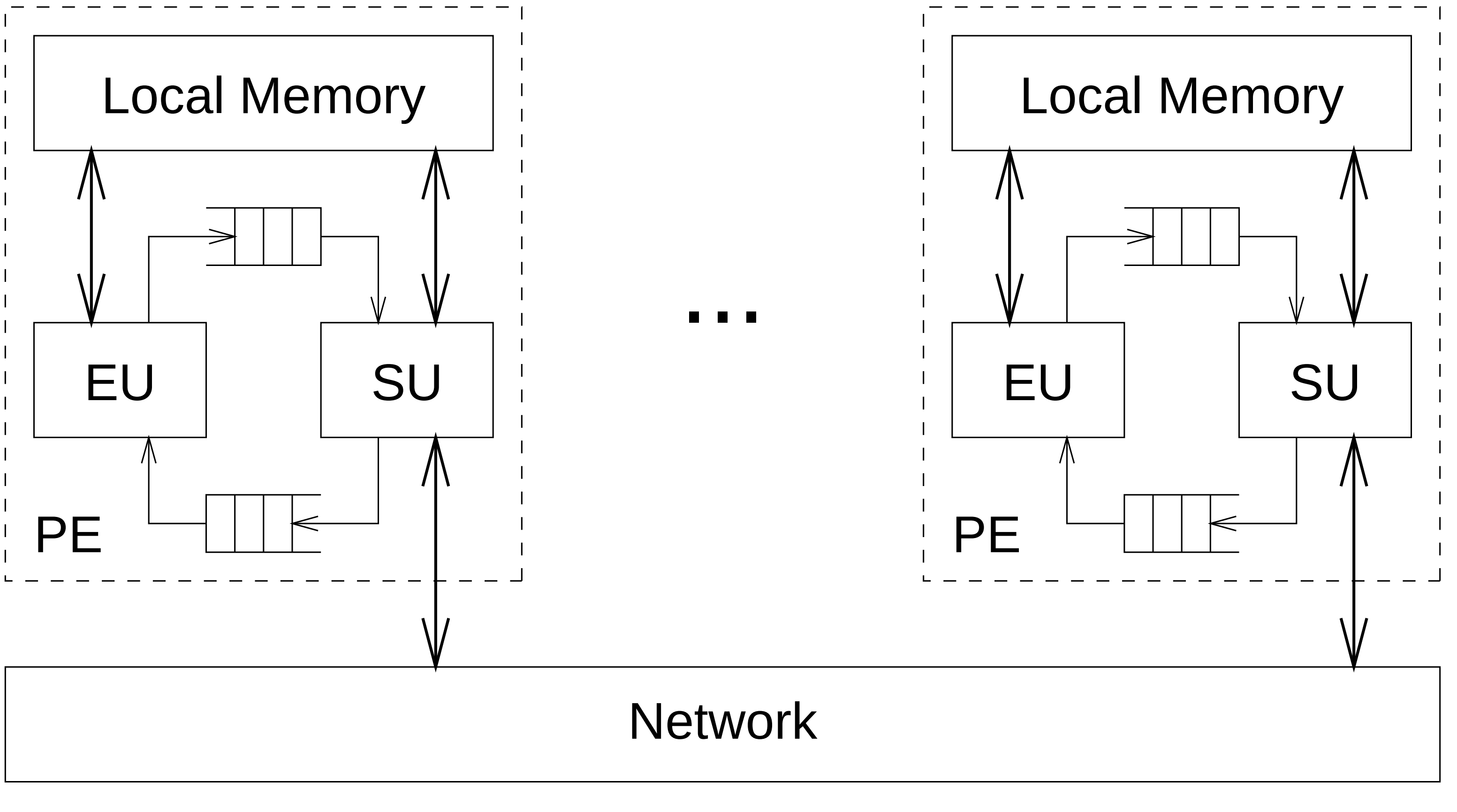
Cost of communication on EARTH-MANNA

EARTH Operation	*Sequential Remote*	*Pipelined Remote*
Read word	7109 ns	1908 ns
Write word	6458 ns	1749 ns
Blkmov	(9600 + n * 25/B) ns	

After communication optimizations

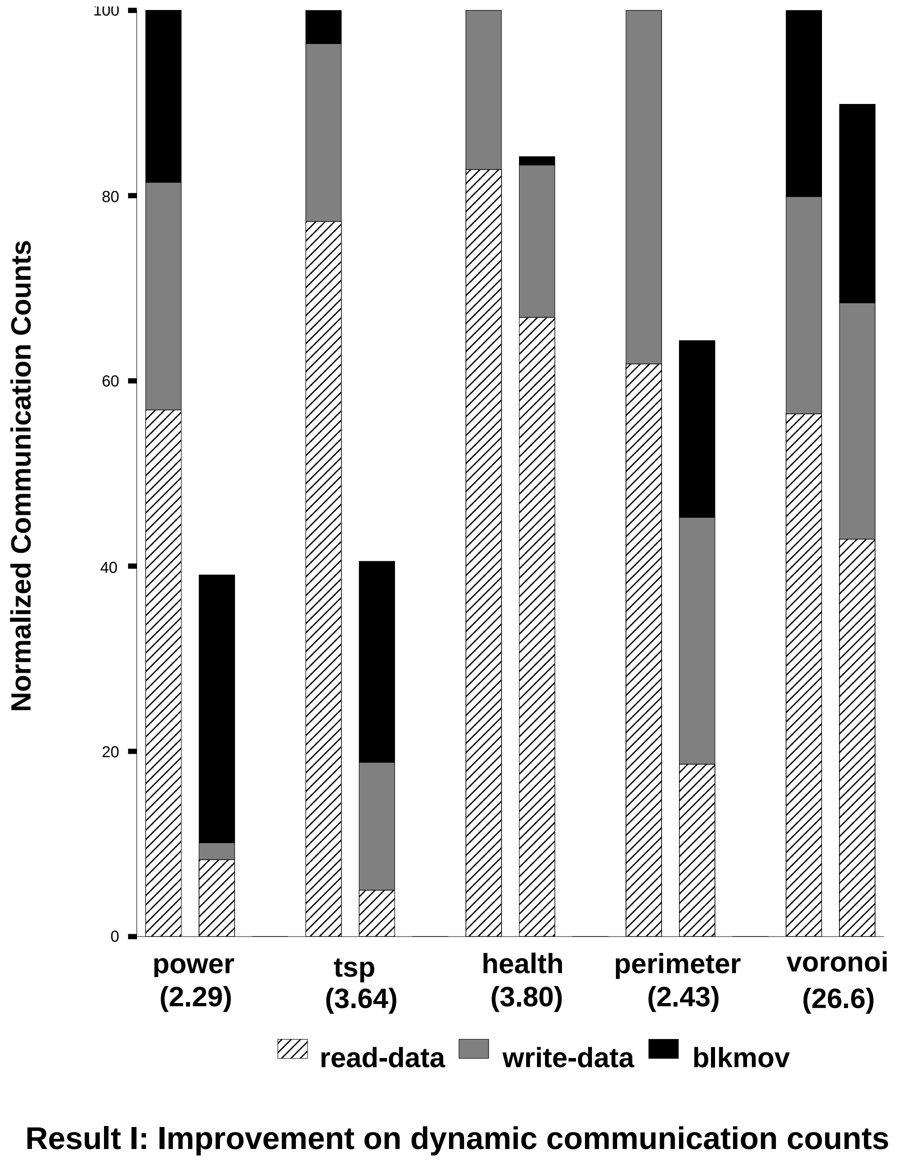
```
S1: l = list->head    ;
S2: blkmov ( l, &comm,  sizeof (Data) )    ;
S3: tmp1 = comm.p;
S4: tmp2 = comm.q;
S5: if  (t != NULL)
S6:    s = t->x    ;
     else
S7:    s = comm.x;
S8: l->q    = tmp2 + s;
```

Testbed: EARTH-MANNA architecture



Benchmark suite

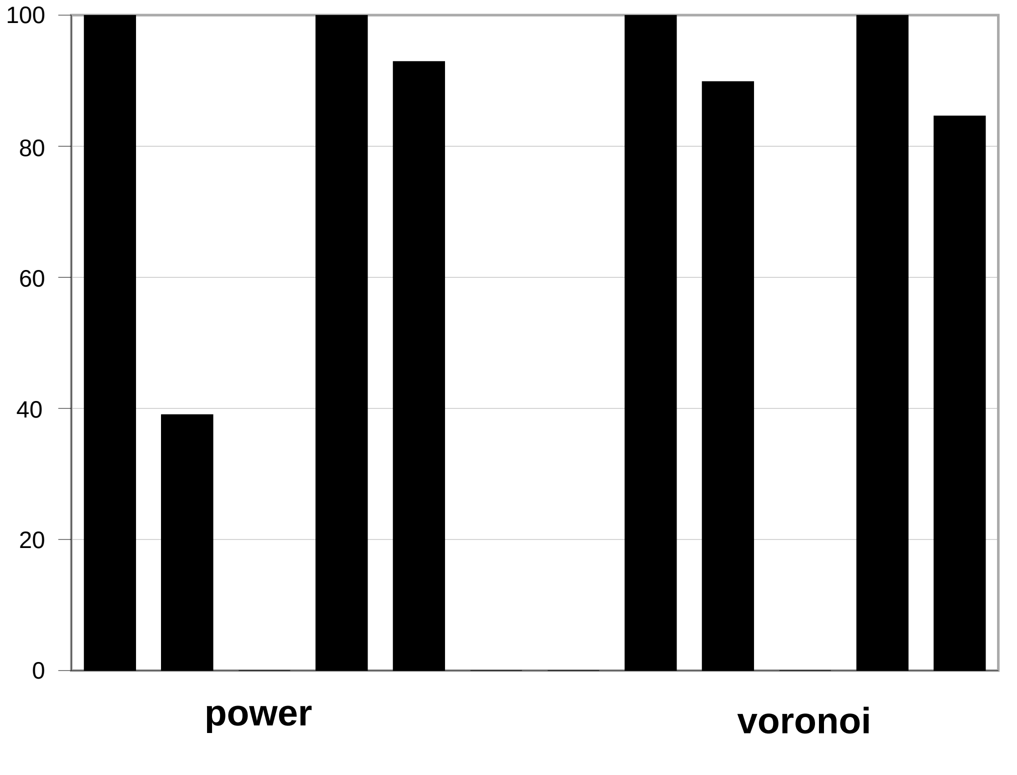
Bchmark	Description
power	Optimization problem based on a k-nary tree
perimeter	Compute the perimeter of a raster image
tsp	Find sub-optimal tour for traveling salesperson problem
health	Simulate a health-care system using a 4-way tree
voronoi	Compute the voronoi diagram of a set of points



Result I: Improvement on dynamic communication counts

Result 2: Performance improvement on Communication Operations

Benchmark		Seq C (sec)	Simple EARTH-C (sec)	Optimized EARTH-C (sec)	Simple Speedup	Optimized Speedup	Optimized vs. Simple (%impr)
power	1	62.4	67.76	66.76	0.92	0.93	**1.48**
	2		35.96	34.41	1.74	1.81	**4.31**
	8		9.77	9.12	6.39	6.84	**6.65**
	16		5.23	4.86	11.93	12.84	**7.07**
tsp	1	23.8	26.95	26.26	0.88	0.91	**2.56**
	2		14.02	13.56	1.70	1.76	**3.28**
	8		4.67	4.29	5.10	5.55	**8.14**
	16		3.77	3.32	6.31	7.17	**11.93**
health	1	143.72	144.09	144.05	0.99	0.99	**0.03**
	2		84.25	80.72	1.71	1.78	**4.19**
	8		42.74	37.69	3.36	3.81	**11.82**
	16		33.88	28.84	4.24	4.98	**14.88**
perimeter	1	5.45	7.32	6.75	0.74	0.81	**7.79**
	2		3.67	3.35	1.49	1.63	**8.72**
	8		1.12	0.98	4.87	5.56	**12.50**
	16		0.75	0.63	7.27	8.65	**16.00**
voronoi	1	3.98	7.27	6.78	0.55	0.59	**6.74**
	2		3.23	2.85	1.23	1.40	**11.76**
	8		1.31	1.17	3.04	3.40	**10.69**
	16		1.04	0.88	3.83	4.52	**15.38**



For each benchmark:

Left two bars: percentage improvement on dynamic counts

Right two bars: percentage improvement on execution time

Related Work

- McGill ongoing research
  - Locality analysis [PACT'97]
  - Heap optimizations for threaded programs [PACT'97][POPL'98]
- Olden's affinity analysis [Carlisle & Rogers, PPoPP'95]
- Parallel program optimizations [Krishnamurthy & Yelick, PLDI'95]
- Synchronization transformations [Diniz & Rinard, POPL'96]
- Software prefetching [Luk & Mowry, ASPLOS'96]
- Optimal code motion [Knoop, Ruthing et.al ToPLAS'94]

Conclusions

- Propose a *communication analysis* framework:
  - possible-placement analysis
  - communication selection

  in order to :
  - move remote read operations as early as possible to overlap between communication and computation,
  - pipeline or block remote reads and writes whenever beneficial.
- Test it on a set of benchmarks, and achieve 2% to 16% improvement.

Future Work

- Study the interaction of communication analysis with locality analysis and thread generation algorithm;
- Add techniques for finding the best organization for fields within each *struct* to minimize the size of blkmovs;