About the Performance of HPF: Improving the Runtime on the Cray T3E with Hardware Specific Properties

Matthias M. Müller
Fakultät für Informatik
Universität Karlsruhe, Germany
muellerm @ ipd.uka.de
http://www.ipd.uka.de/KarHPFn

Common Problem of HPF

- Low performance of compiled programs
- Reason: large architecture idependent communication library
  - Simplifies portability across plattforms
  - But incurs large overhead on parallel execution time
- Result
  - HPF only used for reference implementations and teaching purpose
  - Number cruncher applications are written using MPI or PVM
- Portland Group HPF compiler on Cray T3E no exception to this situation

Suggested Solution

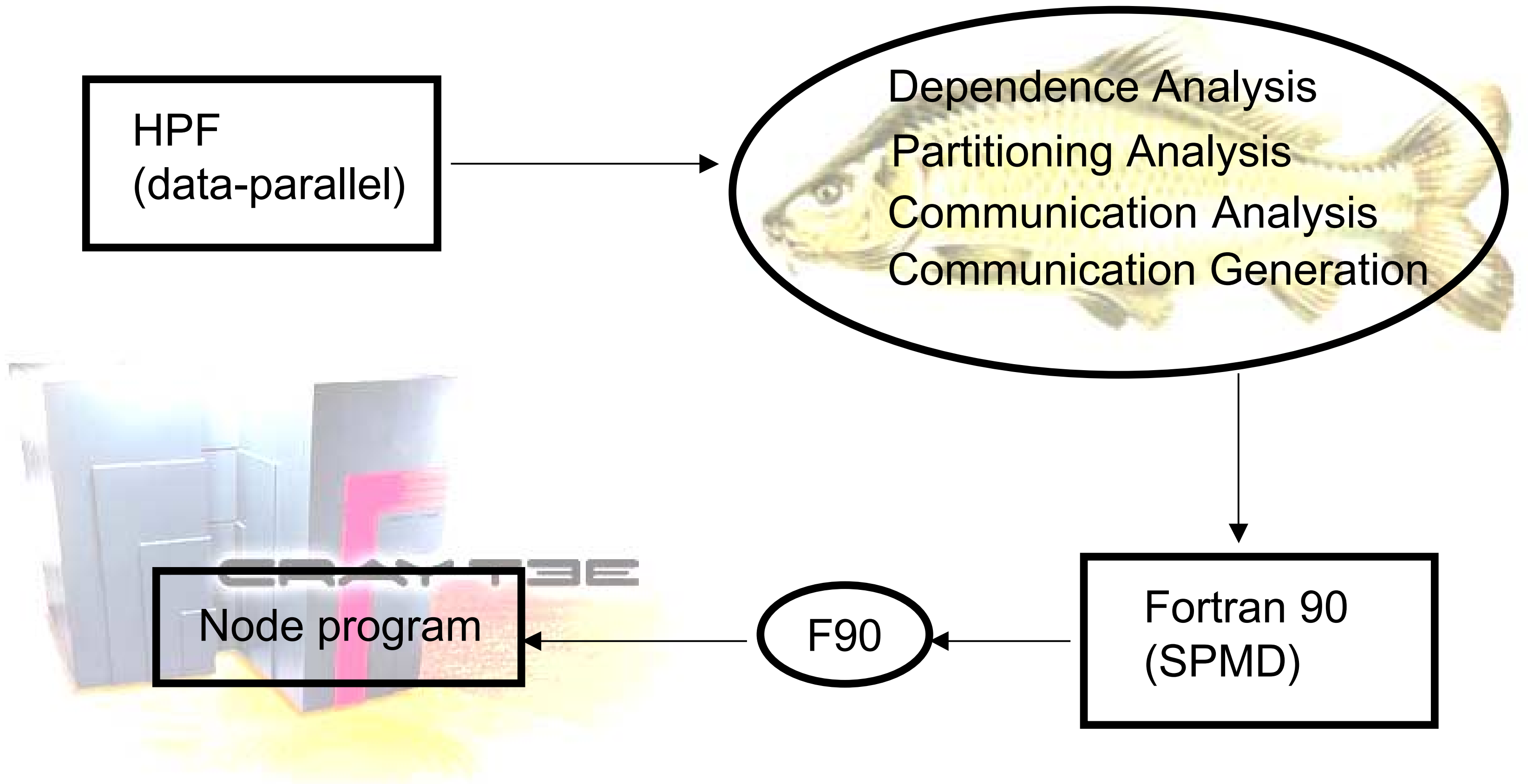
- Take advantage of hardware specific properties of a parallel architecture
- Omit large plattform independent communication library
- Replace it with
  - small highly optimized communication primitives and
  - sophisticated analysis during compilation
- No message passing but prefetching of remote data-elements

Karlsruhe High Performance Fortran

KarHPFn

„Karpfen“ is the german word for carp

Overview


HPF
(data-parallel)
Dependence Analysis
Partitioning Analysis
Communication Analysis
Communication Generation
Fortran 90
(SPMD)
F90
Node program

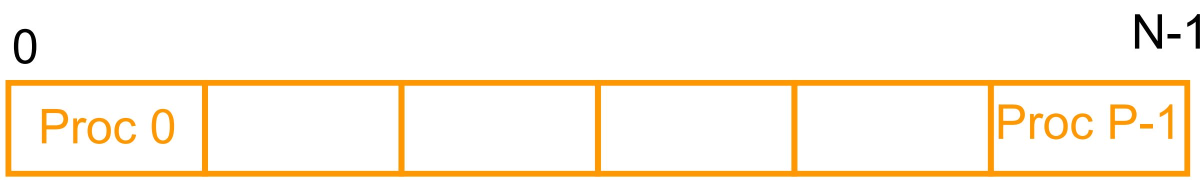
Transformation within KarHPFn

```
!HPF$ DISTRIBUTE (BLOCK) :: A,B,Q
      FORALL i = 0..N-1
        A[i] = B[Q[i]]
      END FORALL
```

- Running example: *Indirect*
- Indirect indexing with permutation Q

1. Step: Virtualization

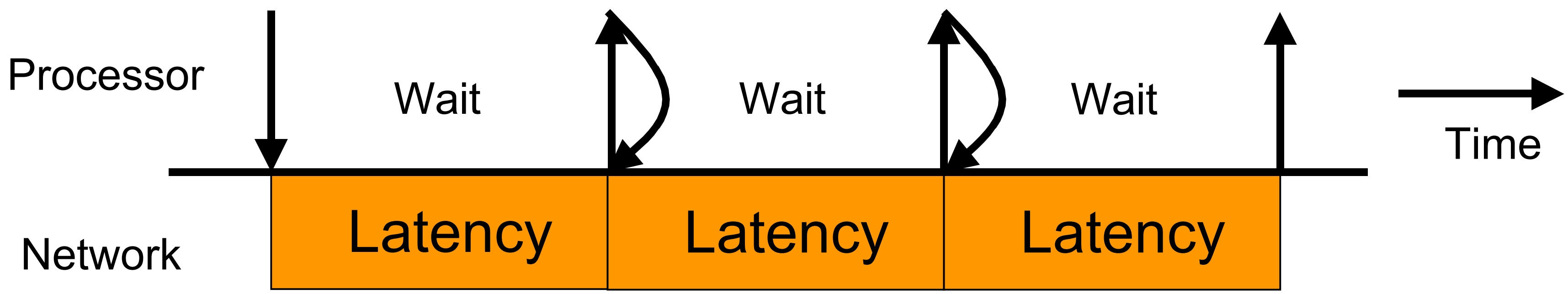
```
V = N/P
FORALL J = 0..P-1 DO
   FOR I=J*V TO (J+1)*V-1 DO
     A[I]=B[Q[I]]
   END
END
```



Virtualization

2. (intermediate) Step: Blocking Communication

```
FORALL J = 0..P-1 DO
  FOR I=J*V TO (J+1)*V-1 DO
    a1=calcAddr(B[Q[I]])
     A[I]=remoteAccess(a1)
  END
END
```



3. Step: Overlapping Communication

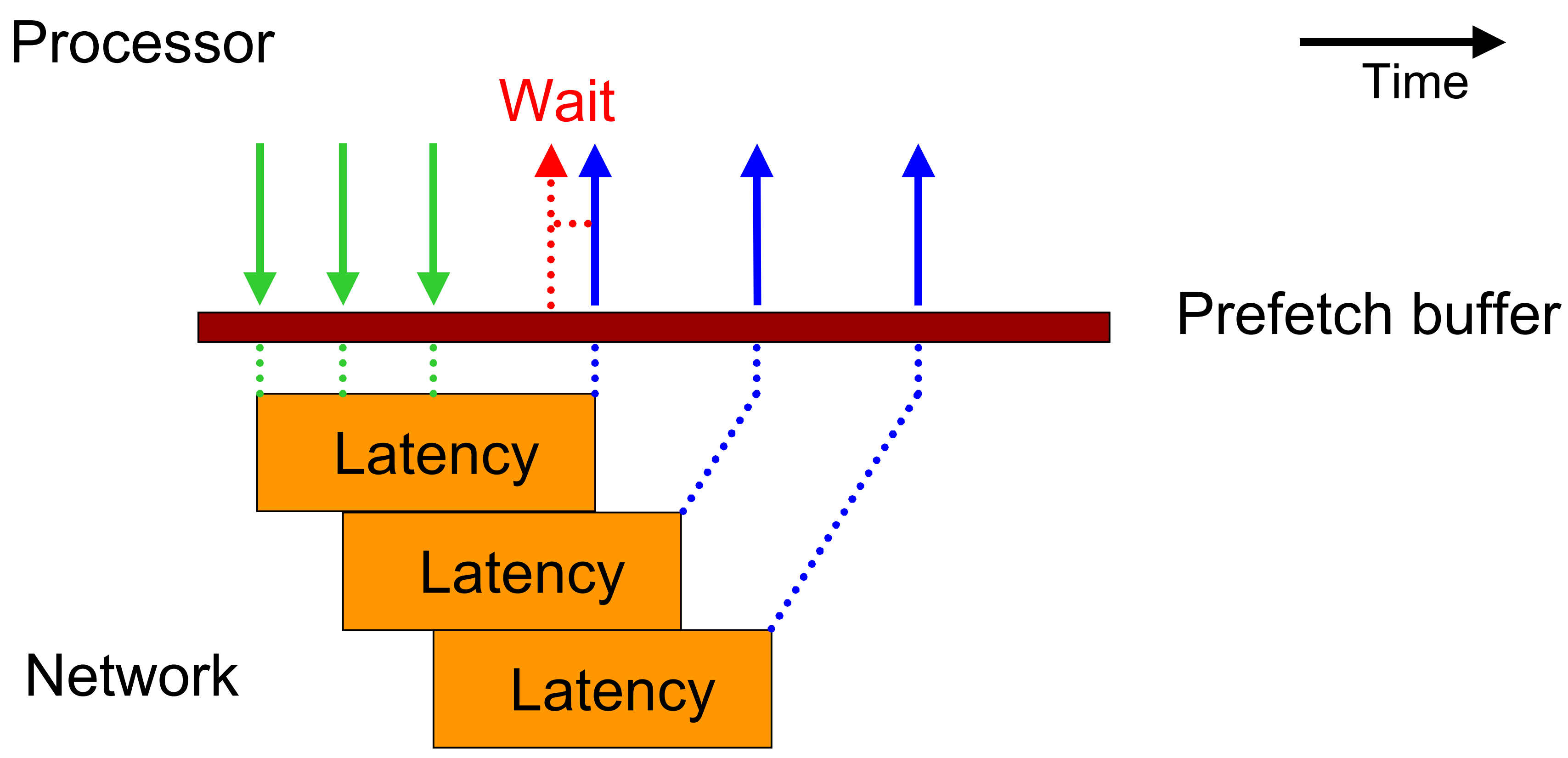
```
FORALL J = 0..P-1 DO
  FOR I=J*V TO (J+1)*V-1 DO
    a1=calcAddr(B[Q[I]])
    prefetch(a1)
  END
  FOR I=J*V TO (J+1)*V-1 DO
    a1=calcAddr(B[Q[I]])
    A[I]=access(a1)
  END
END
```

Prefetch Loop

Access Loop

Combined Access & Prefetch Loop

3. Step: Overlapping Communication (cont.)



3. Step: Overlapping Communication (cont.)

- KarHPFn uses vector operations for prefetch and access
- Vector operations reduce overhead for prefetch and access
- Prerequisite: displacements of elements are equidistant and known at compile-time
- Otherwise: prefetching with element-wise operations
- Vector length on Cray T3E $L=8$

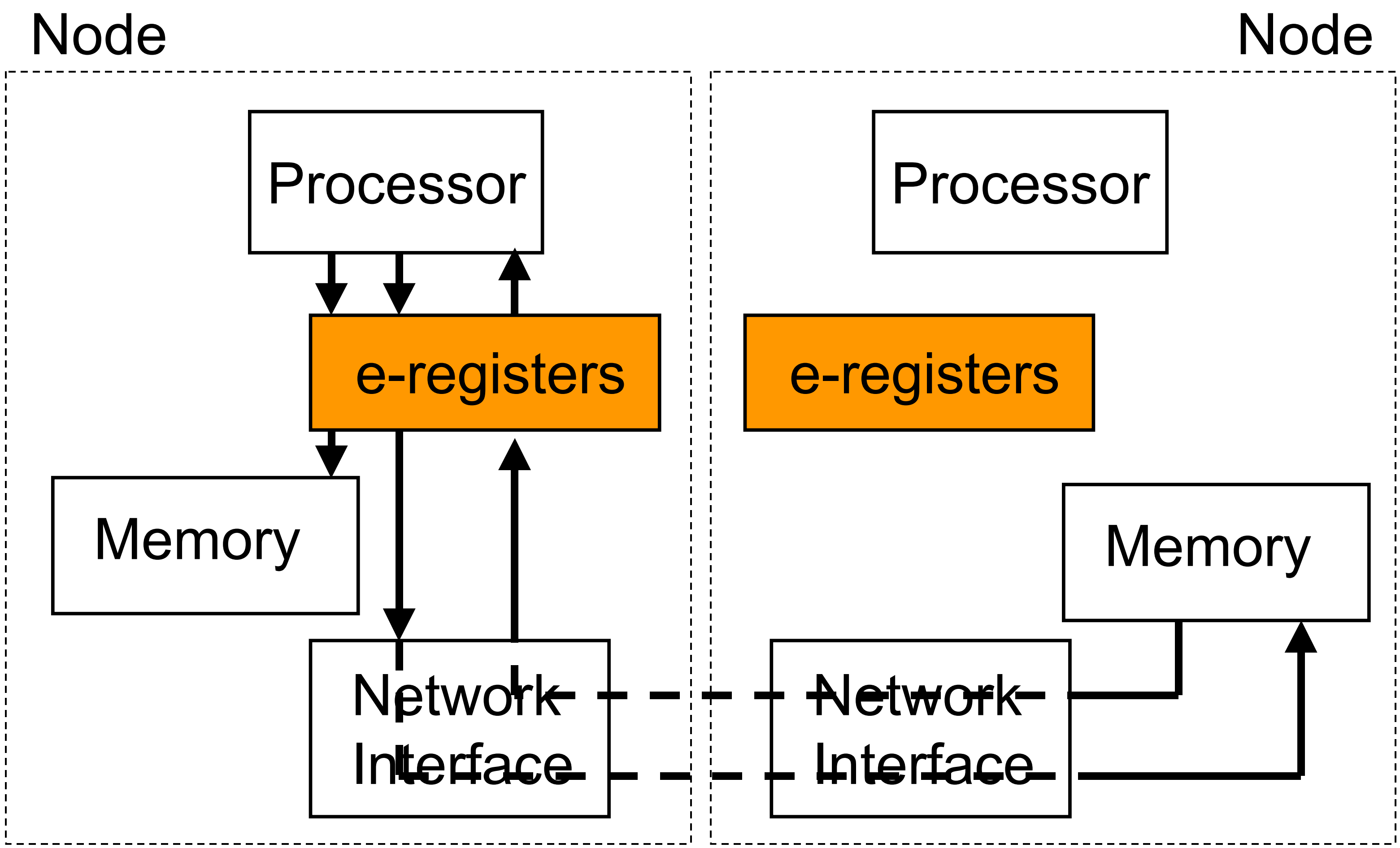
E-Registers

- 512 e-registers overall, 480 for free usage
- Lie in I/O-space of processor
- Bypass processor cache
- Only means for communication in Cray T3E
- [Scott96] 128 e-registers suffice to hide network latency and to get maximum throughput
- Size of prefetch-buffer 128 e-registers or 16 vectors

E-Registers (cont.)

- Synchronization of processor and e-registers done in hardware
- Access of local and remote memory without disturbing processor execution
- Address translation can be swapped out to e-register logic: *Hardware Centrifuge*
  – Further decrease of communication overhead
  – Only possible with special problem sizes
- Example: HWC of Rotate

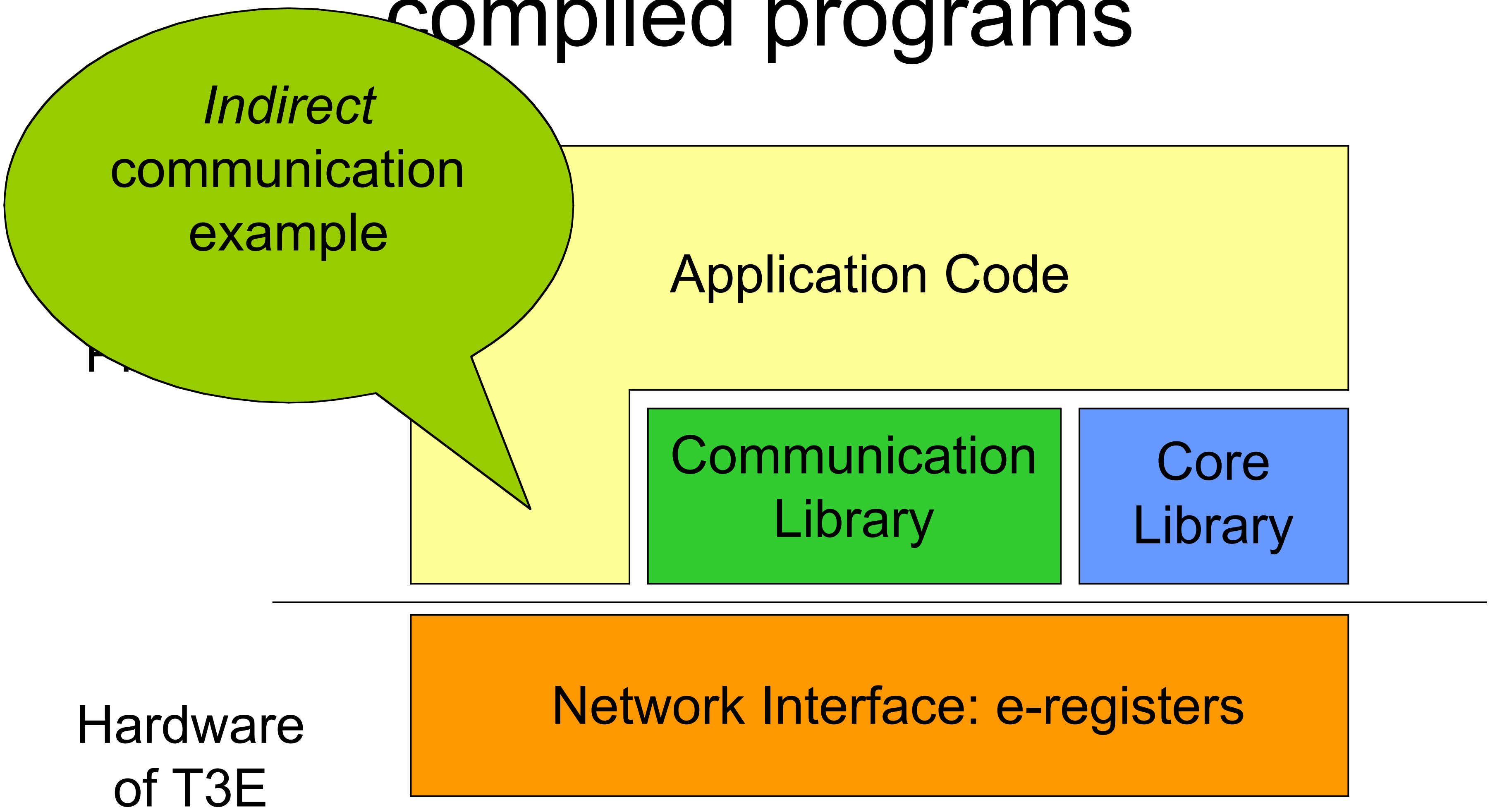
Communication on Cray T3E



4. Step: Code Generation

```
        low$I = MAX(DAD$X%Low(1),1)
        high$I = MIN(DAD$X%High(1),LENGTH)
        inc$I = 1
        EMobe(397) = LOC(X(0))
!DIR$ SUPPRESS Emobe
        tmp$1 = ISHFT(99,MPC_BS_EDATA_MOBE)
        tmp$1 = OR(tmp$1,ISHFT(MY_PE(),MPC_BS_DFLTCENTPE))
        EMobe(400) = DAD$Y%CentMask
!DIR$ SUPPRESS Emobe
        EMobe(401) = LOC(Y(0))
!DIR$ SUPPRESS Emobe
        EMobe(398) = ISHFT(DAD$X%SliceProd(1),3)
!DIR$ SUPPRESS Emobe
        EMobe(402) = ISHFT(DAD$Y%SliceProd(1),3)
!DIR$ SUPPRESS Emobe
        PrefRegs = 0
        DO pre$I=low$I-DAD$X%Low(1),low$I-DAD$X%Low(1)+
                (MIN(128,high$I-DAD$X%Low(1)-(low$I-DAD$X%Low(1))+1)-1)
         ! Procs Y 1
         tmp$2 = ISHFT((V(pre$I)-1)/DAD$Y%Slice(1)*map$LINE(1),
                    MPC_BS_DFLTCENTPE)
         ! addr Y 1
         tmp$3 = MOD(V(pre$I)-1,DAD$Y%Slice(1))
         ERegGet(PrefRegs) = OR(DAD$Y%Mobe,
                                OR(tmp$2,ISHFT(tmp$3,3)))
!DIR$ SUPPRESS ERegGet
         PrefRegs = PrefRegs+1
        END DO
        Vecs = 0
        tmp$4 = low$I-DAD$X%Low(1)+(CEILING(REAL(high$I-DAD$X%Low(1)-
                      (low$I-DAD$X%Low(1))+1)/8)*8-129)
        DO access$I=low$I-DAD$X%Low(1),tmp$4,8
         ! addr X 1
         tmp$5 = access$I
          VecPut(Vecs) = OR(tmp$1,ISHFT(tmp$5,3))
!DIR$ SUPPRESS VecPut
          Vecs = AND(Vecs+1,31)
          tmp$6 = MIN(high$I-DAD$X%Low(1),pre$I+7)
          DO pre$I=pre$I,tmp$6
           ! Procs Y 1
           tmp$2 = ISHFT((V(pre$I)-1)/DAD$Y%Slice(1)*map$LINE(1),
                      MPC_BS_DFLTCENTPE)
           ! addr Y 1
           tmp$3 = MOD(V(pre$I)-1,DAD$Y%Slice(1))
           ERegGet(PrefRegs) = OR(DAD$Y%Mobe,
                      OR(tmp$2,ISHFT(tmp$3,3)))
!DIR$ SUPPRESS ERegGet
           PrefRegs = AND(PrefRegs+1,255)
          END DO
         END DO
         call MEMORY_BARRIER ()
         DO access$I=access$I,high$I-DAD$X%Low(1)-7,8
          ! addr X 1
          tmp$5 = access$I
          VecPut(Vecs) = OR(tmp$1,ISHFT(tmp$5,3))
!DIR$ SUPPRESS VecPut
          Vecs = AND(Vecs+1,31)
         END DO
         Regs = Vecs*8
         DO access$I=access$I,high$I-DAD$X%Low(1),1
          ! addr X 1
          tmp$5 = access$I
          ERegPut(Regs) = OR(tmp$1,ISHFT(tmp$5,3))
!DIR$ SUPPRESS ERegPut
          Regs = AND(Regs+1,255)
         END DO
         call ERegWait ()
```

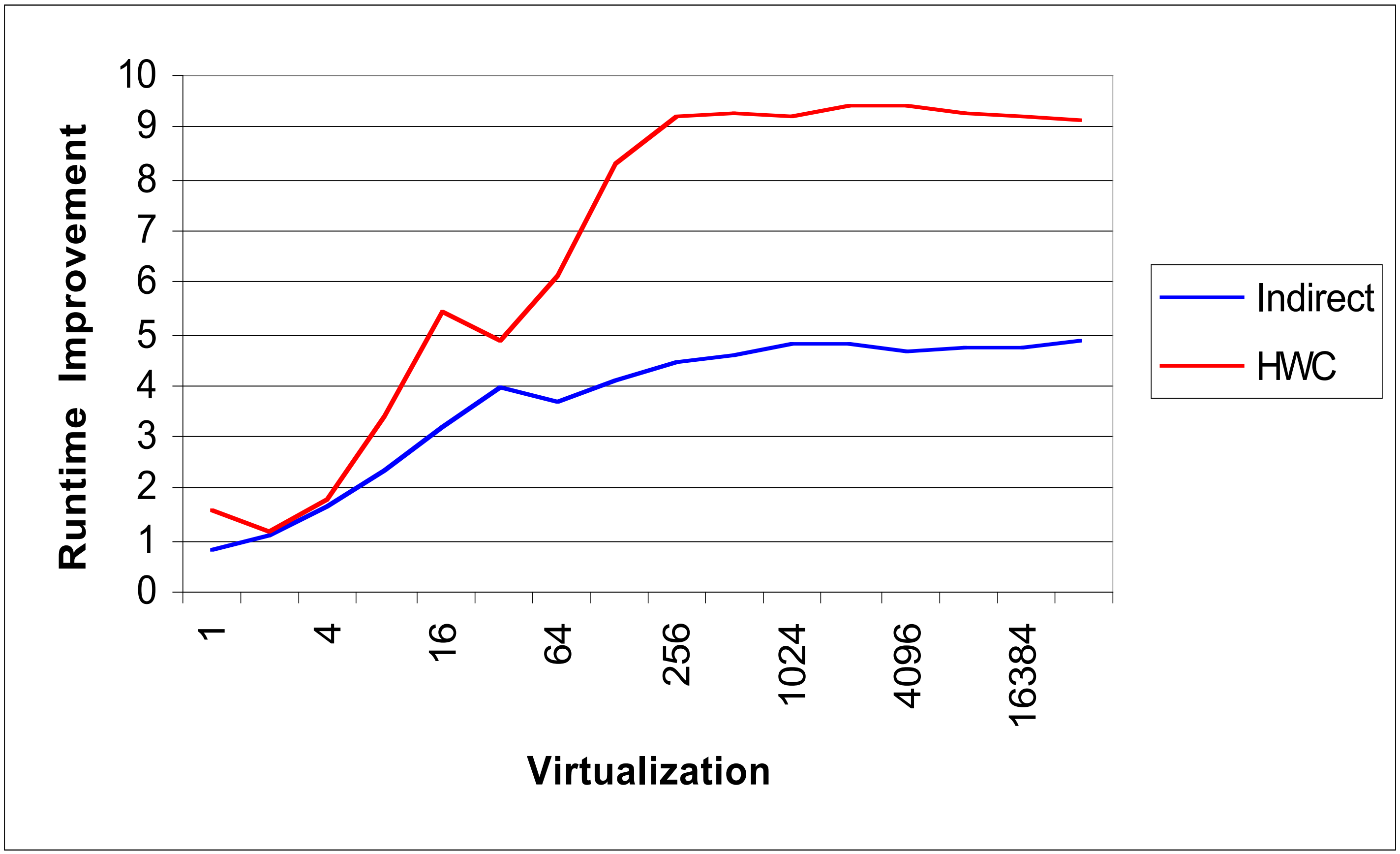
Structure of KarHPFn compiled programs



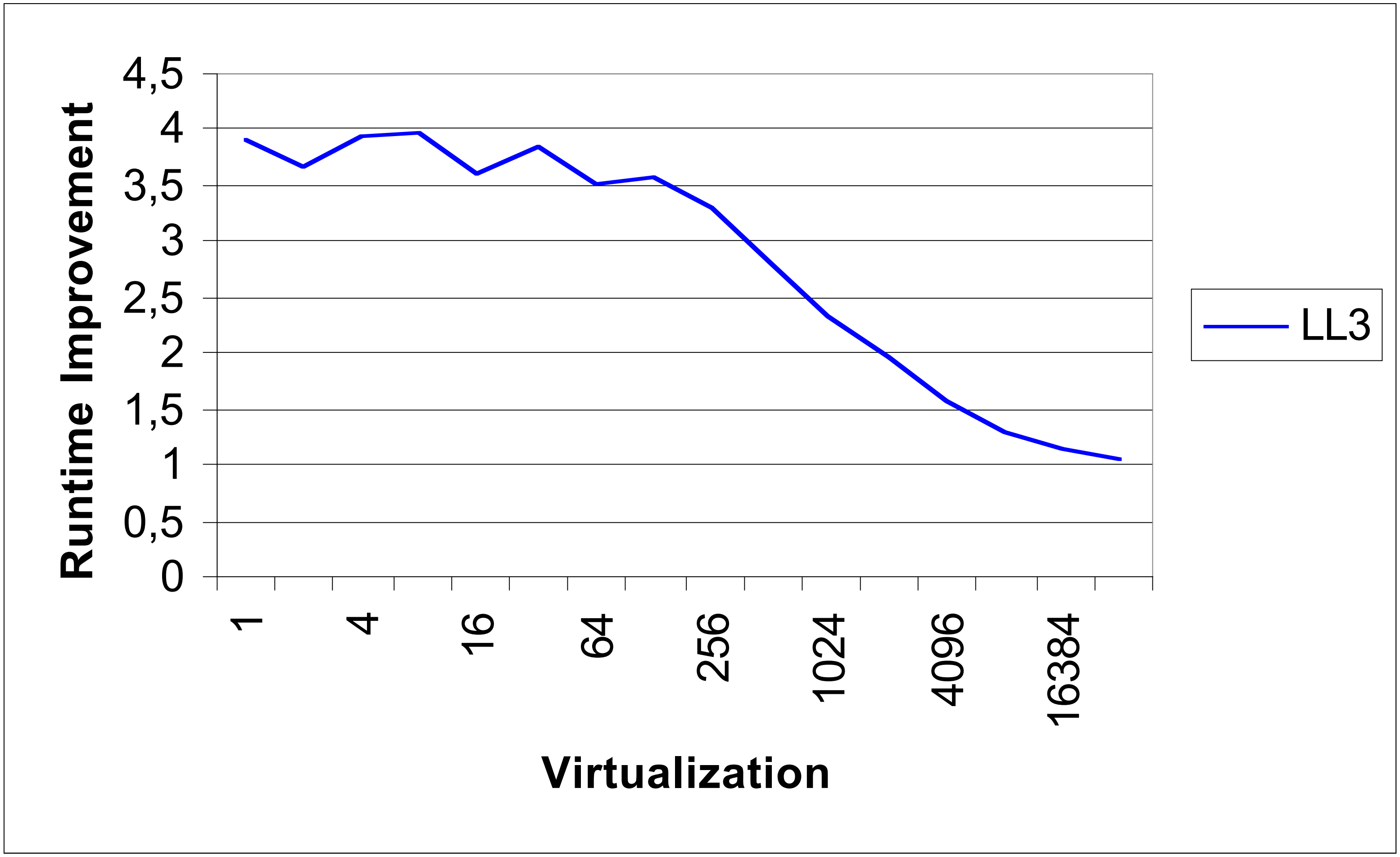
Benchmark Set

- 25 Benchmarks
- From wide range of common parallel algorithm classes
- Measurements on 32 and 128 processors
- Comparison of relative runtime improvement of KarHPFn compiled programs to PGI HPF compiled programs
- Target architecture: T3E-900

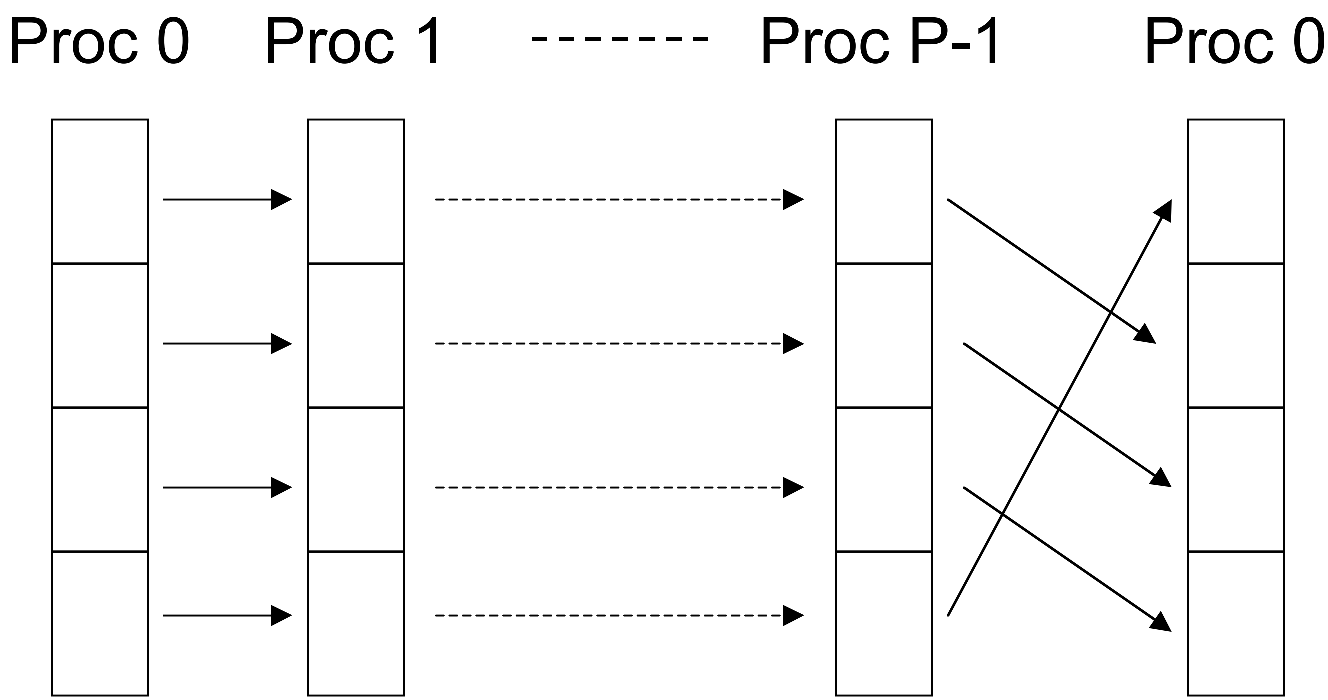
Running Example: Indirect on 32 Processors



Scalar-Product: LL3 on 32 Processors

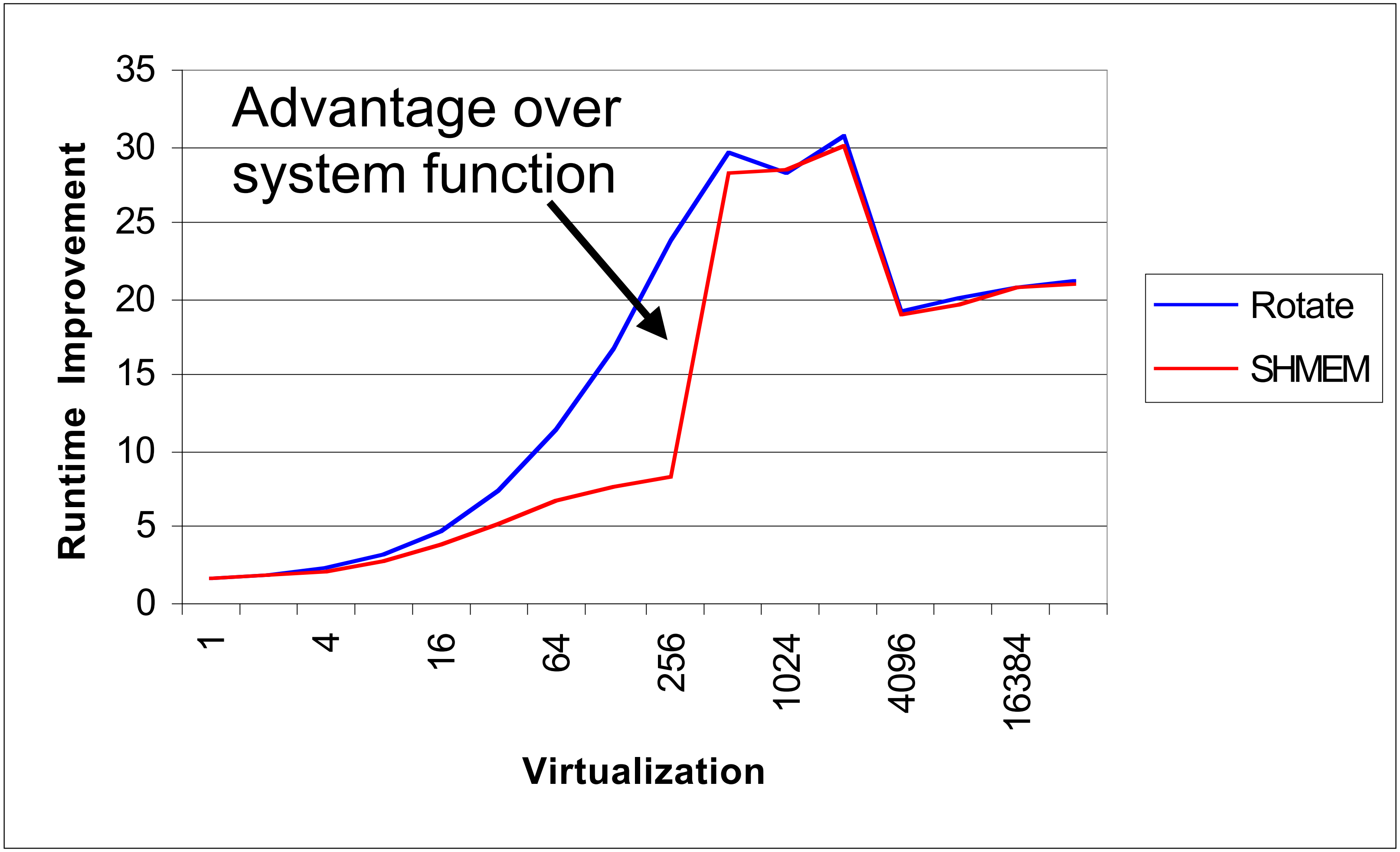


Communication Pattern of Rotate cshift

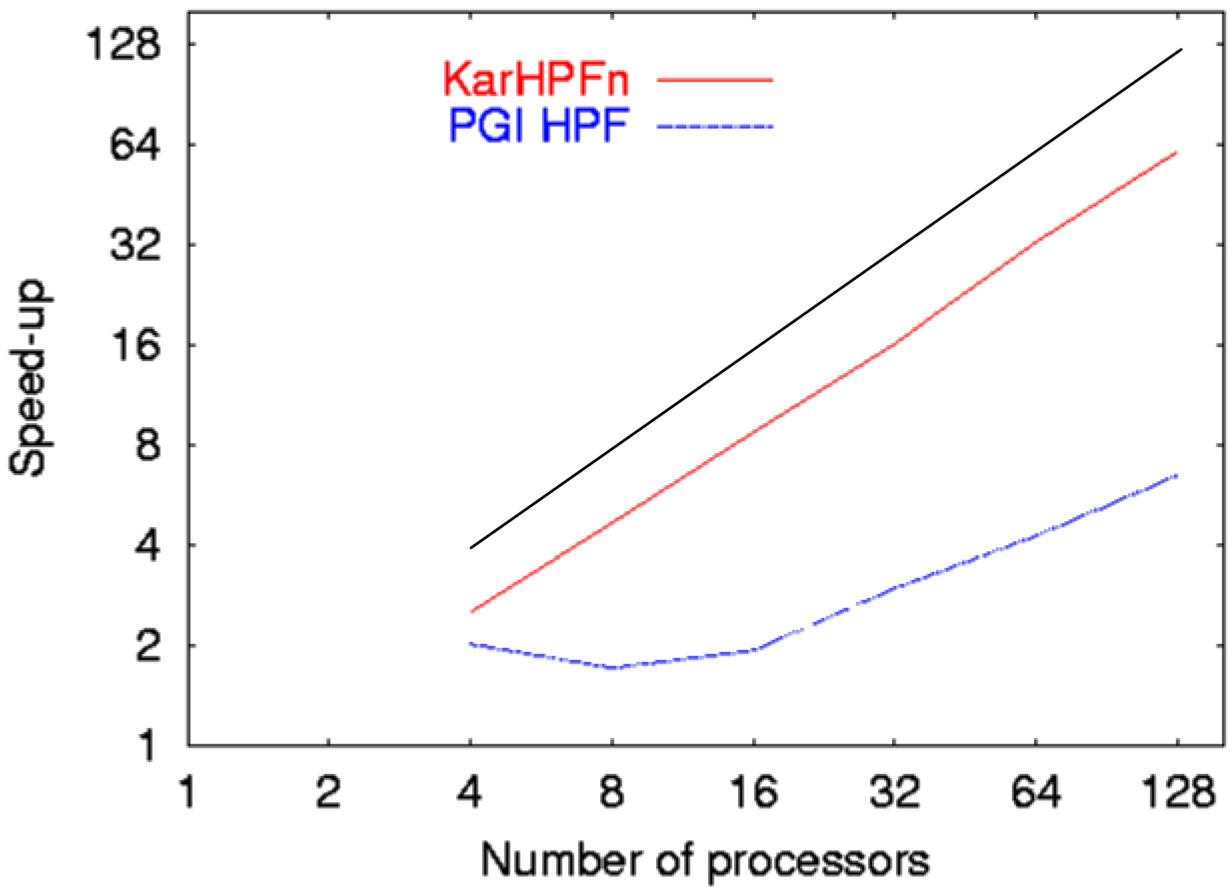


CYCLIC distribution

Rotate: cshift on 32 Processors

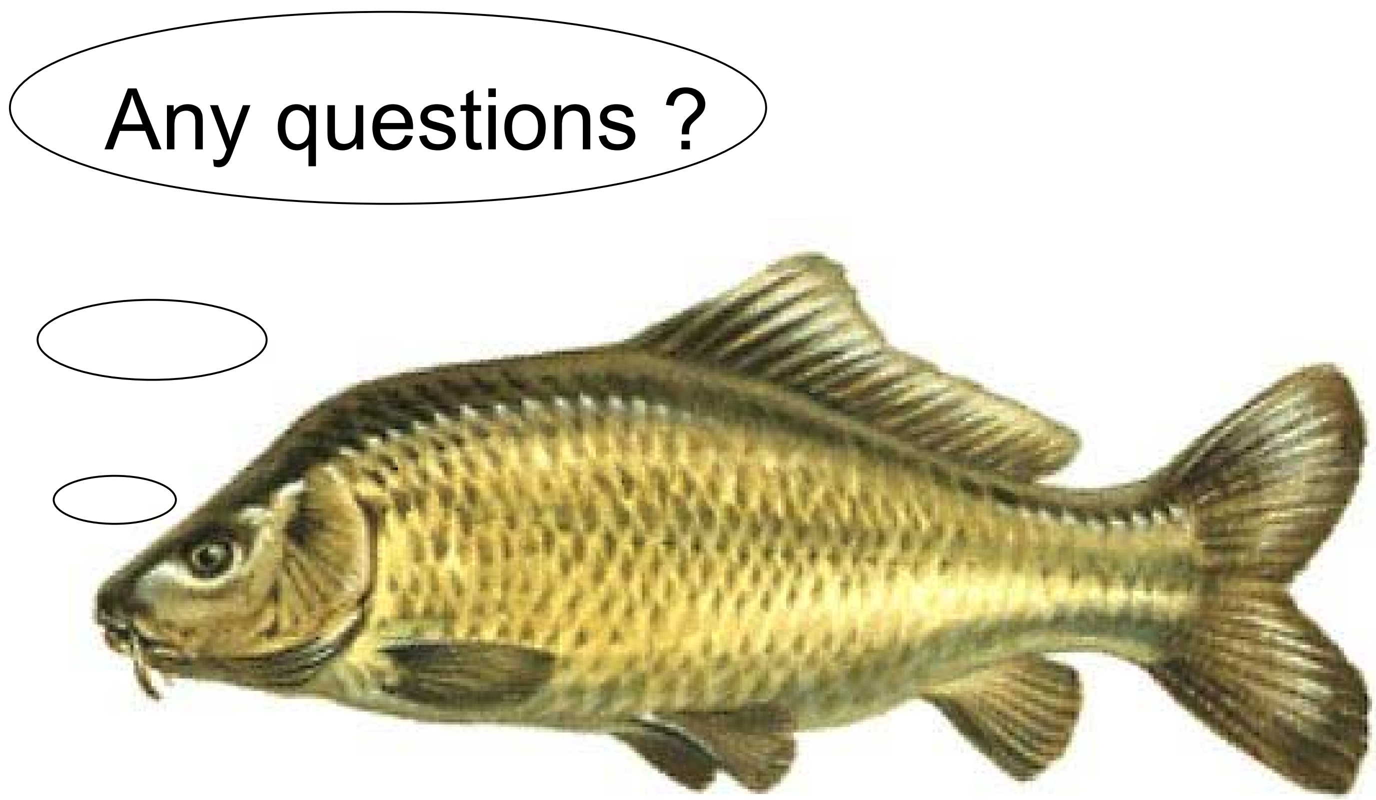


Speed-up for PDE1



Summary

- All KarHPFn compiled programs are faster than PGI HPF compiled programs
  - Up to 30 times faster for $V \geq 8$
  - Up to 1700 times faster for $1 \leq V \leq 4$
- Speed up on 128 processors
  - KarHPFn between 64 and 79
  - PGI HPF between 7 and 33


Any questions ?