Sparse Approximate Inverse
Based on Processor Virtualization

Kai Wang
Parallel Programming Laboratory
Department of Computer Science
University of Illinois At Urbana-Champaign

Contents

- Introduction
- Preconditioning
- Sparse Approximate Inverse Preconditioning
- Virtualization
- SAI based on virtualization

Introduction Cont.

What is our problem?

In scientific and engineering applications, we often need to slove

- A partial Differential Equation (PDE)
- Discretize to get a matrix A, here A is a $n * n$ matrix and
  - Sparse: Very few nonzero elements
  - Large: Millions of unknowns
- Solve the large sparse linear system

$$Ax = b$$

A small sparse matrix (5point)

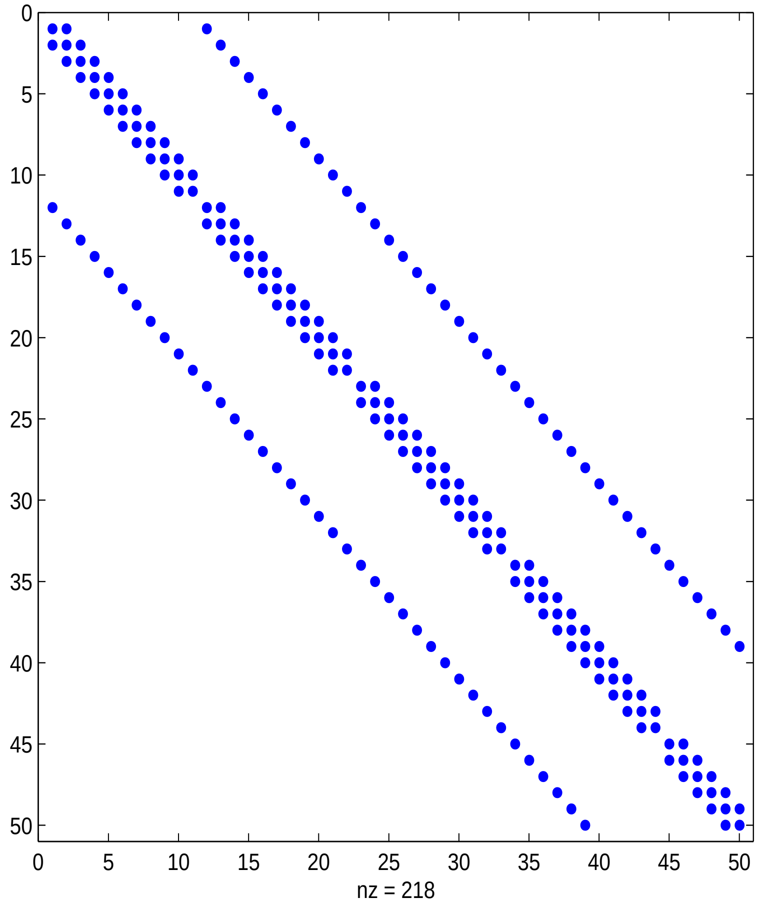


Figure 1: The structure of a small sparse matrix, $\mathbf{50 * 50}$.

Introduction Cont.

How to solve this sparse linear system $Ax = b$?

- When A is a small matrix, Gaussian Eliminations are enough.
- However when A is very large, Gaussian Eliminations are not applicable, since their
  - $O(n^3)$ computational cost
  - $O(n^2)$ memory cost
  - Difficult to implement on parallel computers

  $n = 10^6$, computational cost $= O(10^{18})$ and memory cost $= O(10^{12})$.

Introduction Cont.

Research interest:

Solve this linear system

- Low memory and computational cost (efficiency)
- Robustness (effectiveness)
- On parallel platforms (Parallelism)

Solution: Iterative methods

Iterative Methods

Basic iterative methods

- Jacobi, Gauss-Seidel, and SOR

They all have the form

$$x_{k+1} = Gx_k + f$$

- Advantage
  - Good parallelism (Matrix Vector Product)
- Disadvantages
  - Not robust
  - Slow convergence

Iterative Methods Cont.

Krylov subspace methods

- GMRES, BCG ...
  - Good parallelism
  - More robust

These methods may still

- Fail to converge
- Slow convergence

for ill-conditioned matrices.

Solution: Change to a good-conditioned matrix

Iterative Methods Cont.

Preconditioned Krylov subspace methods

- **Setup phase:** Find a matrix $\boldsymbol{N}$ and transform the linear system into an equivalent one

$$\boldsymbol{NAx} = \boldsymbol{Nb}$$

  - $\boldsymbol{N}$ should be computed cheaply ($<<$ the cost of $\boldsymbol{A}^{-1}$)
  - $\boldsymbol{NAx} = \boldsymbol{Nb}$ is easier to solve than $\boldsymbol{Ax} = \boldsymbol{b}$ (better-conditioned)

  Here $\boldsymbol{N}$ is called the preconditioner for $\boldsymbol{A}$.

- **Solve phase:** Solve the transformed system by Krylov subspace methods (GMRES algorithm, BCG algorithm)

Current Status

Two typical preconditioning techniques to compute N

- ILU
  - Form: $N = (LU)^{-1}$, $LU \approx A$
  - Transformed equation:

$$(LU)^{-1}Ax = (LU)^{-1}b$$

- Sparse approximate inverse (SAI)
  - Form: $N = M$, $M \approx A^{-1}$
  - Transformed equation:

$$MAx = Mb$$

SAI Preconditioners

Goal: $M \approx A^{-1}$, $AM \approx I$

- Frobenius norm minimization:

$$\begin{aligned} \|AM - I\|_F^2 &= \sum_{k=1}^{n} \|(AM - I)e_k\|_2^2 \\ &= \sum_{k=1}^{n} \|Am_k - e_k\|_2^2 \\ & \quad M = (m_1, m_2, \ldots\ldots, m_n) \end{aligned}$$

- We have n independent minimization problems

$$\min_{m_k} \|Am_k - e_k\|_2, \quad k = 1, 2, \ldots, n$$

Good parallelism

SAI Preconditioners Cont.

Exact or Approximate Inverse

- Without any constraint on m_k, solving each minimization problem

  $$\min_{m_k} \|Am_k - e_k\|_2, \quad k = 1, 2, \ldots, n$$

  will equal to solving the original problem $Ax = b$

  - this would be impractical (CPU & memory)

Solution: Guess a sparsity pattern (nonzero positions) for M

SAI Preconditioners Cont.

- Suppose we are given a sparsity pattern for M
  - Only limited nonzeros in each column $\tilde{m}_k$
- For each k, construct $\tilde{A}_k$, corresponding to $\tilde{m}_k$
- The individual minimization becomes

$$\min_{\tilde{m}_k} \|\tilde{A}_k \tilde{m}_k - \tilde{e}_k\|_2$$

- $\tilde{A}_k$ is rectangular, and has full rank (if A is nonsingular)

QR Factorization

- We can perform a QR factorization on $\tilde{A}_k$ (small)

$$\tilde{A}_k = Q_k \begin{pmatrix} R_k \\ 0 \end{pmatrix}$$

  - Q_k is orthogonal, $Q_k^T Q_k = I$
  - R_k is nonsingular upper triangular

- Compute

$$\tilde{c}_k = Q_k^T \tilde{e}_k$$

- and solve

$$\tilde{m}_k = R_k^{-1} \tilde{c}_k$$

An Algorithm of SAI

Algorithm **0.1** *Construct a static pattern sparse approximate inverse preconditioner.*

```
1.  Given a drop tolerance $\epsilon$
2.  Sparsify $A$ with respect to $\epsilon$
3.  Compute a sparse approximate inverse $M$ according to
    the sparsity pattern of $A$
4.  Drop small entries of $M$ with respect to $\epsilon$
5.  $M$ is the preconditioner for $Ax = b$
```

- Matrix A is partitioned and distributed row by row
- The nonzero position of A (sparsified) is the sparsity pattern.
- If preconditioner is not good enough, use the nonzero pattern of A^2, or A^3 as the sparsity pattern

Processor Virtualization

Standard MPI programming model

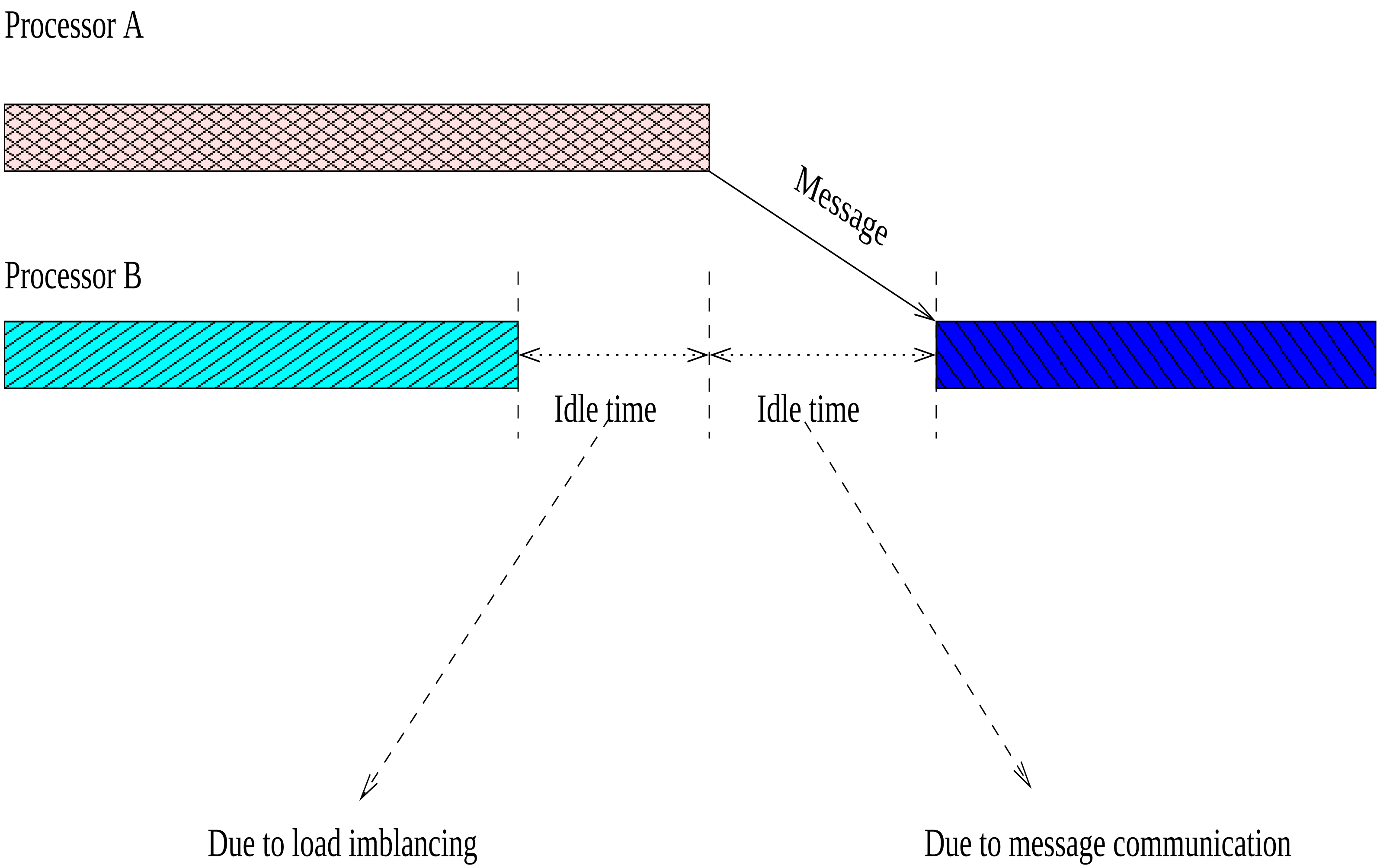
- To run on P processors
- Divide the computation into P processes
- Each for one physical processors
  - The division is not natural
  - Significant effort is required for load balance

Processor Virtualization

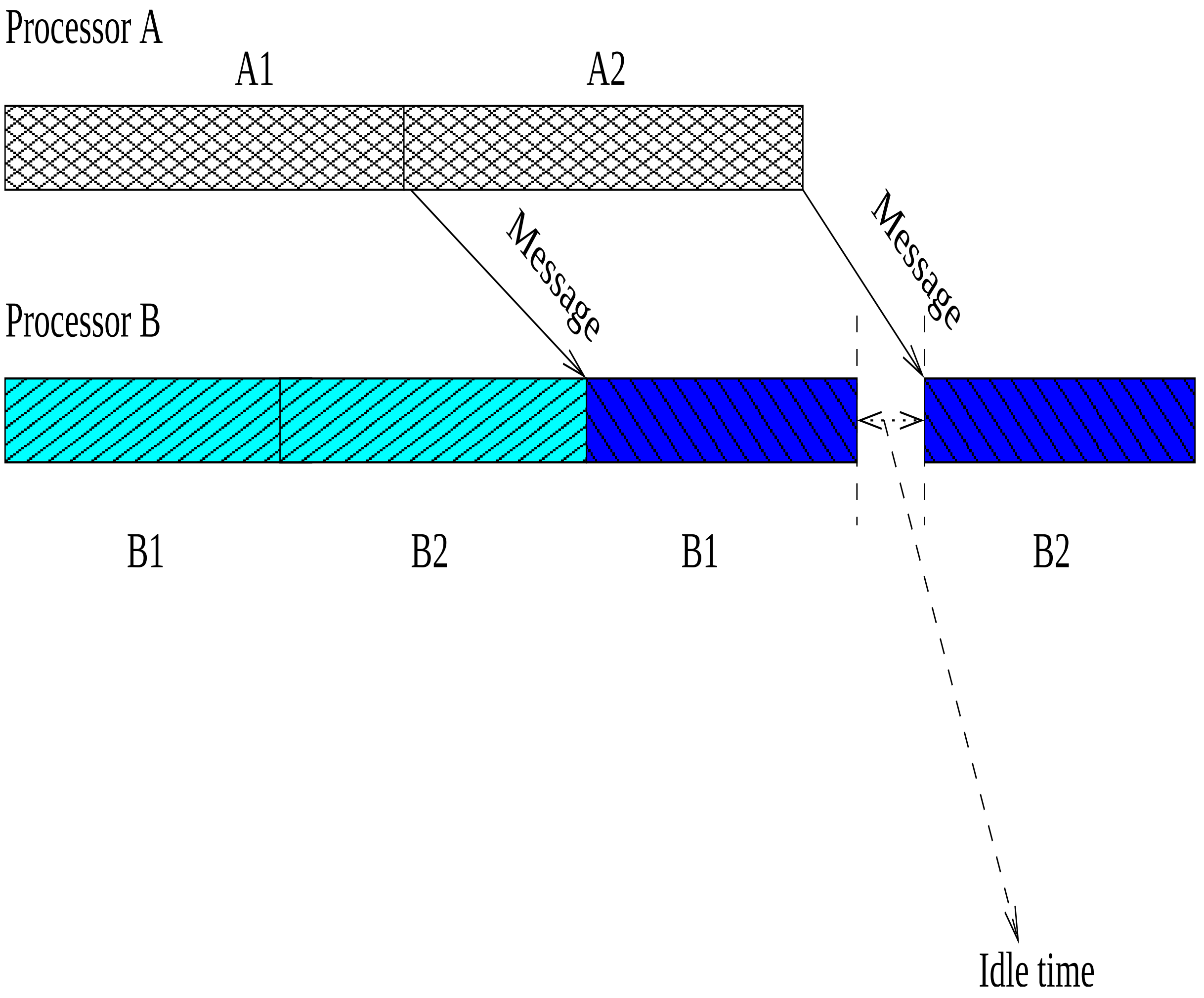
Virtualization

- To run on P processors
- Divide the computation into any number of processes independent of P
- Runtime system maps those processes to physical processors
  - Automatic load balancing
  - Cache performance
  - Adaptive overlapping of communication and computation

An example


Processor A
Message
Processor B
Idle time
Idle time
Due to load imblancing
Due to message communication

An example


Processor A
A1
A2
Message
Message
Processor B
B1
B2
B1
B2
Idle time

SAI based on Processor Virtualization

- Load balancing is not a serious problem
  - Give each physical processor roughly the same number of rows
- Cache performance is important for the implementation
- Adaptive overlapping of communication and computation makes efficient implementation. Save CPU time.

SAI based on Processor Virtualization

Can the computation and communication be overlapped?

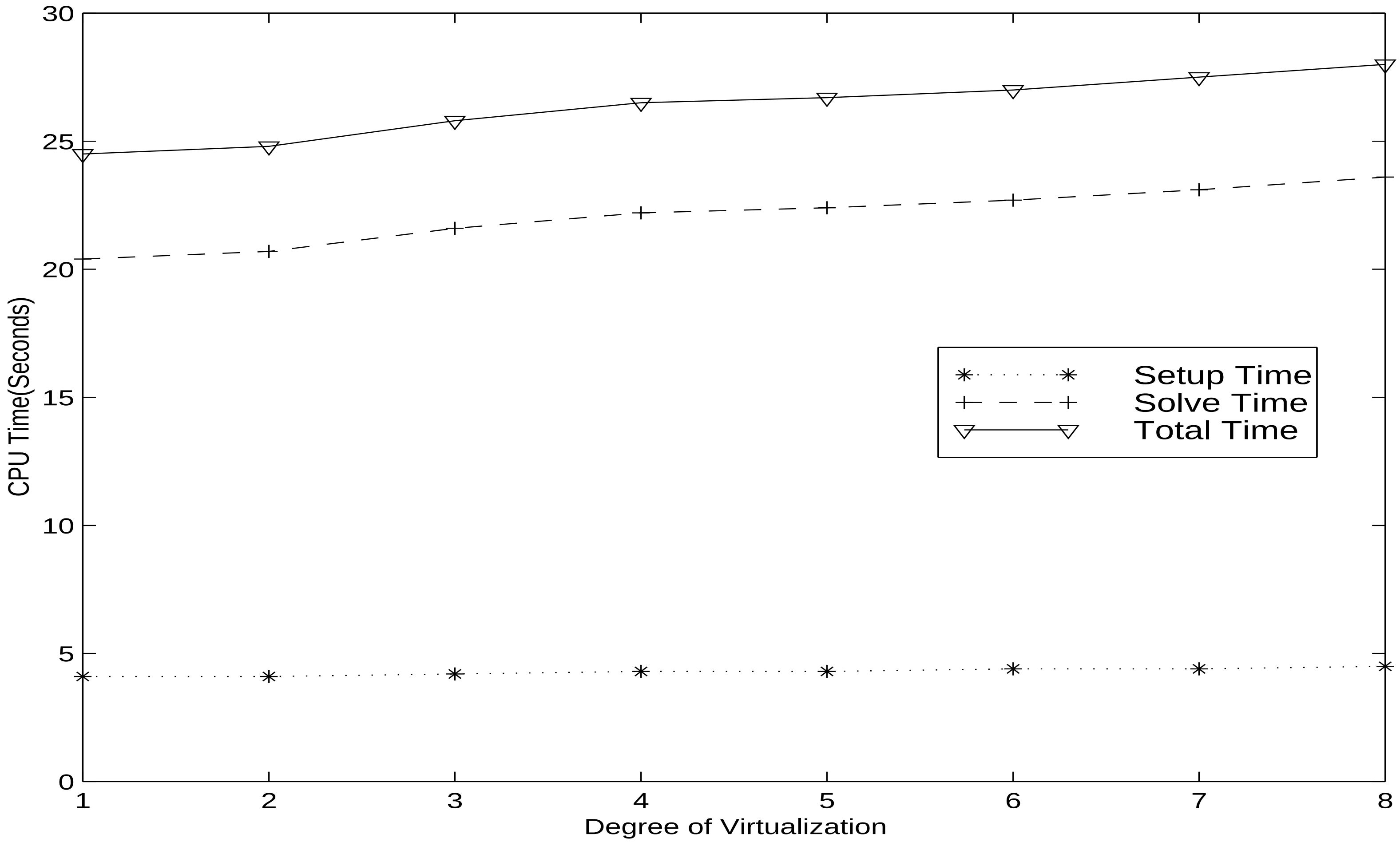
- Matrix vector product is the main computation
- Each virtual processor need the vector store in other processors
- If dense matrix, this is an all to all communication. No overlapping
- Sparse matrix has limited nonzeros
  - Communication happens in a subset
  - Can overlap

Numerical Results

- Implementation is based ParaSails of Edmond. Chow
  - It uses static sparsity pattern
  - Use Lapack and Blas to deal with cache performance
- Use AMPI
- Run on Tungsten
- Solving $\mathbf{3}$D convection-diffusion equations.
- $\mathbf{1000}$ iterations.

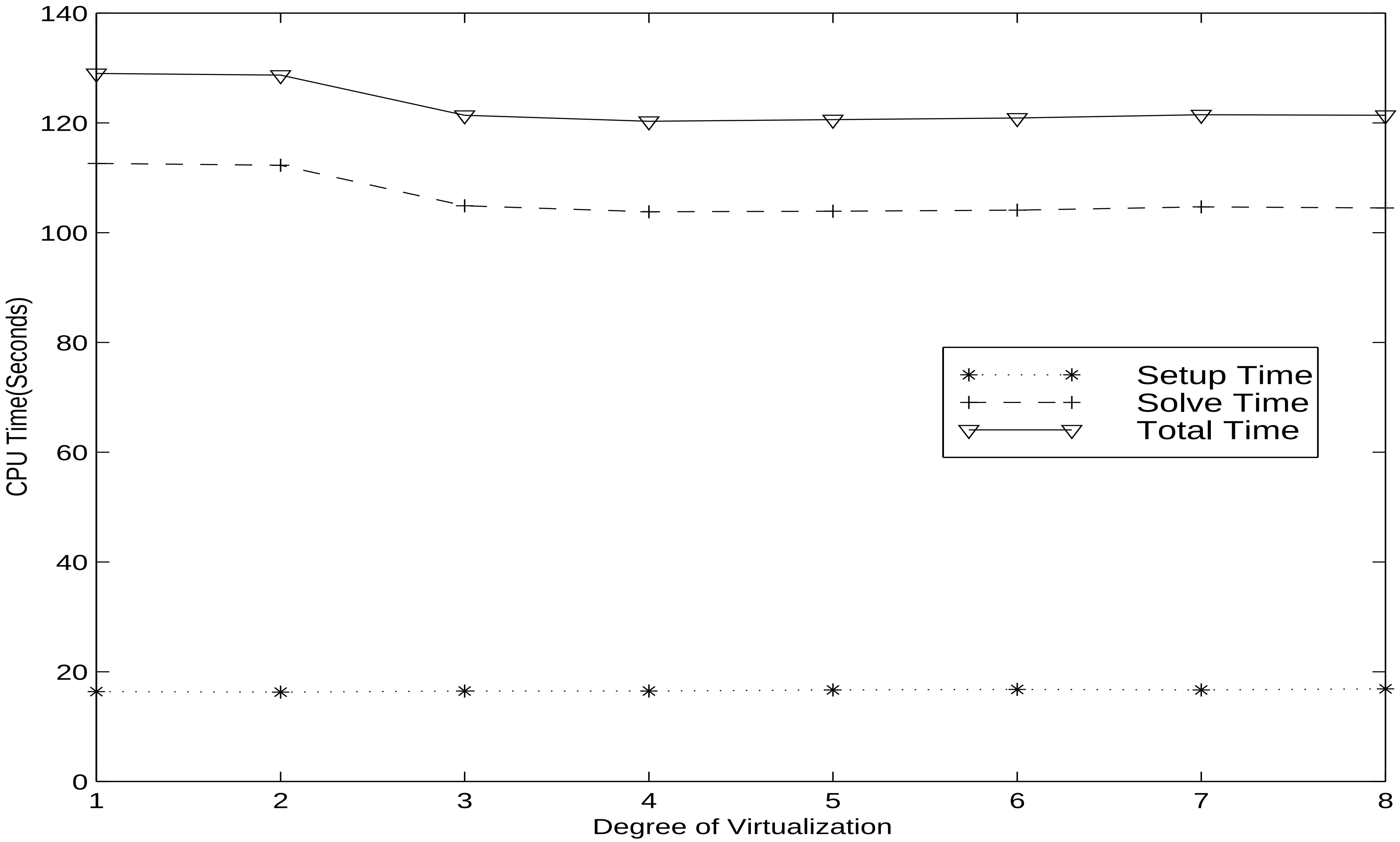
Virtualization Overhead

One physical processor. Number of unknowns=40000.



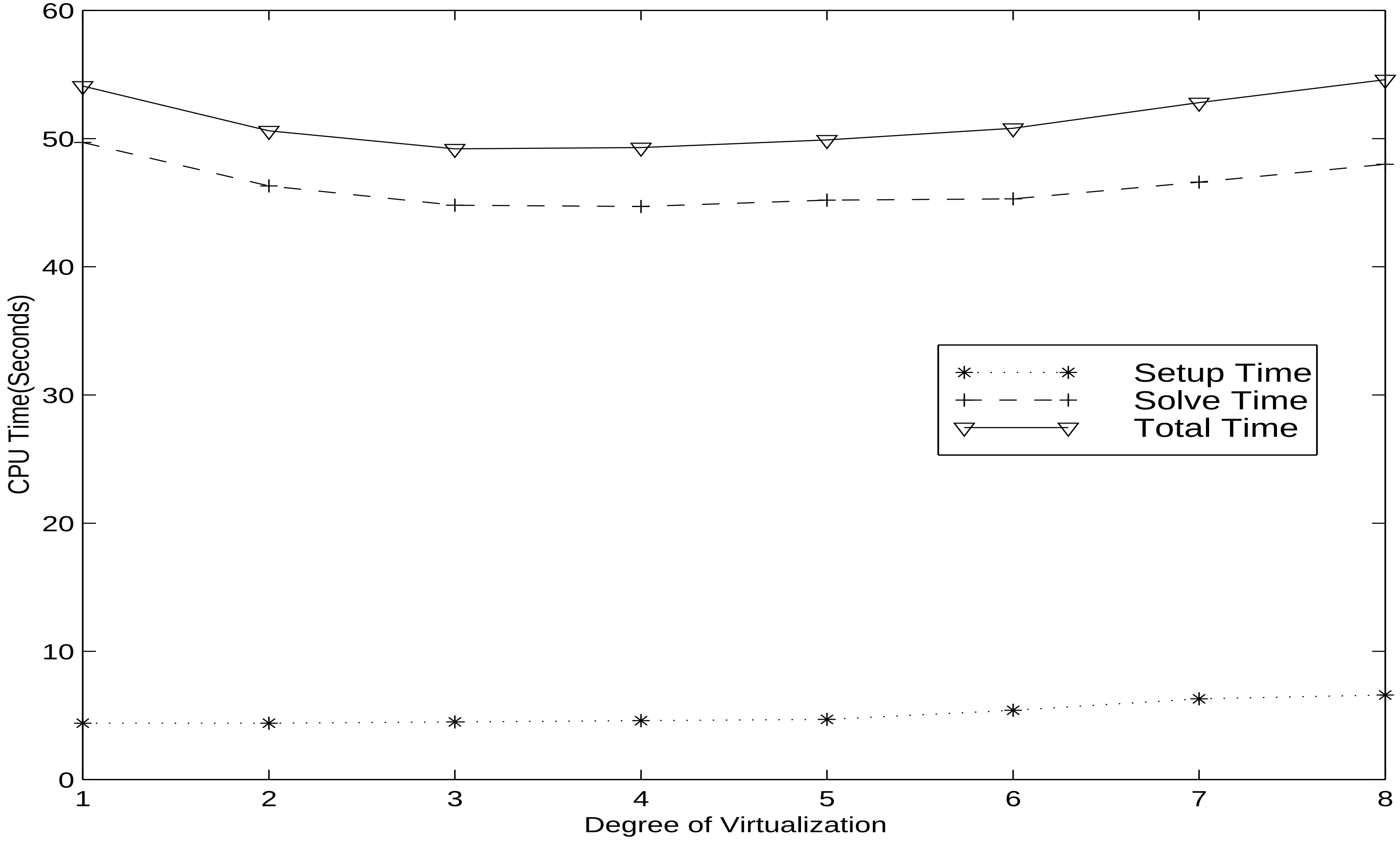
Cache Performance

One physical processor. Number of unknowns=**160000**.



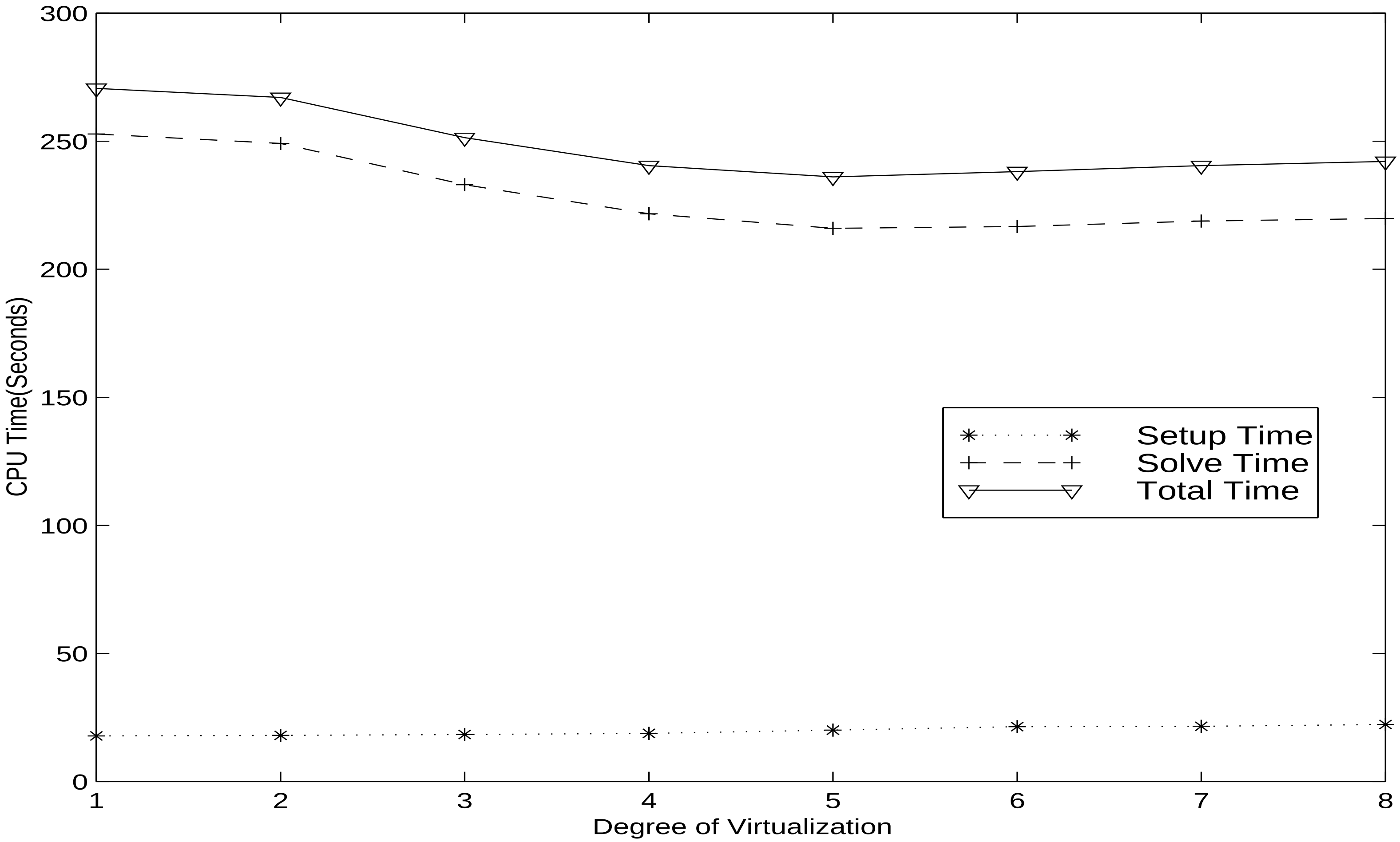
Adaptive Overlapping

16 physical processor. Number of unknowns=**640000**.



Virtualization Overhead

32 physical processor. Number of unknowns=**5120000**.



Conclusion

- Show the performance of SAI preconditioning when using processor virtualization.
- Speedup in solving phase
  - Benefits from cache performance
  - Benefits from adaptive overlapping
- No speedup in setup phase
  - Setup phase has good cache management and few communications.
  - Speedup may be expected in setup phase of multilevel or multistep preconditioning.