A nonblocking approach for the parallel computation of the Laplacian

Julio Sánchez-Curto and Pedro Chamorro-Posada
Departamento de teoría de la señal, comunicaciones e ingeniería telemática
E.T.S.I. Telecomunicación
Universidad de Valladolid, Spain

ISPDC 2022: 21^{th} International Symposium on Parallel and Distributed Computing

Outline

Outline

1. Motivation of our work

Outline

1. Motivation of our work
2. Classical approach

Outline

1. Motivation of our work
2. Classical approach
3. The nonblocking alternative

Outline

1. Motivation of our work
2. Classical approach
3. The nonblocking alternative
4. Performance tests and results

Outline

1. Motivation of our work
2. Classical approach
3. The nonblocking alternative
4. Performance tests and results
5. Discussion

Outline

1. Motivation of our work
2. Classical approach
3. The nonblocking alternative
4. Performance tests and results
5. Discussion
6. Conclusions

Motivation of our work

Motivation of our work

- The Laplacian is a widely used operator in physics

Motivation of our work

- The Laplacian is a widely used operator in physics
- It is usually embedded in Partial Differential Equations (PDEs)

Motivation of our work

- The Laplacian is a widely used operator in physics
- It is usually embedded in Partial Differential Equations (PDEs)
- Spectral methods constitute a way of solving PDEs

Motivation of our work

- The Laplacian is a widely used operator in physics
- It is usually embedded in Partial Differential Equations (PDEs)
- Spectral methods constitute a way of solving PDEs
- If Fourier Series are used, the Laplacian can be computed based on the differentiation property

$$\nabla^2 u(x, y) \stackrel{\mathfrak{F}}{\longleftrightarrow} -(\Omega_1^2 + \Omega_2^2) U(\Omega_1, \Omega_2)$$

Motivation of our work

- The Laplacian is a widely used operator in physics
- It is usually embedded in Partial Differential Equations (PDEs)
- Spectral methods constitute a way of solving PDEs
- If Fourier Series are used, the Laplacian can be computed based on the differentiation property

$$\nabla^2 u(x,y) \overset{\mathfrak{F}}{\longleftrightarrow} -(\Omega_1^2 + \Omega_2^2)U(\Omega_1,\Omega_2)$$

In terms of parallel computing,

- Straightforward strategy: use of parallel FFTs routines

Motivation of our work

- The Laplacian is a widely used operator in physics
- It is usually embedded in Partial Differential Equations (PDEs)
- Spectral methods constitute a way of solving PDEs
- If Fourier Series are used, the Laplacian can be computed based on the differentiation property

$$\nabla^2 u(x,y) \overset{\mathfrak{F}}{\longleftrightarrow} -(\Omega_1^2 + \Omega_2^2) U(\Omega_1, \Omega_2)$$

In terms of parallel computing,

- Straightforward strategy: use of parallel FFTs routines
- We, however, consider the whole kernel

Motivation of our work

- The Laplacian is a widely used operator in physics
- It is usually embedded in Partial Differential Equations (PDEs)
- Spectral methods constitute a way of solving PDEs
- If Fourier Series are used, the Laplacian can be computed based on the differentiation property

$$\nabla^2 u(x, y) \overset{\mathfrak{F}}{\longleftrightarrow} -(\Omega_1^2 + \Omega_2^2)U(\Omega_1, \Omega_2)$$

In terms of parallel computing,

- Straightforward strategy: use of parallel FFTs routines
- We, however, consider the whole kernel
- It exploits the peculiar features of the Laplacian operator to get a more efficient parallel implementation

Classical approach

Classical approach

Use of state-of-the-art parallel FFTs routines, based on the transpose method

Classical approach

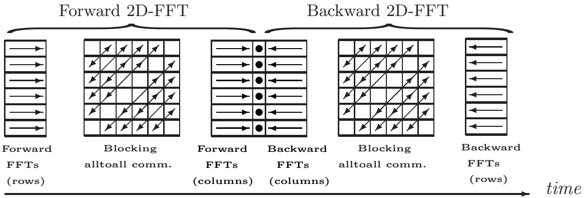
Use of state-of-the-art parallel FFTs routines, based on the transpose method
For the two dimensional (2D) case:

$$y[n_1, n_2] = \frac{1}{N} \sum_{k_2=0}^{N_2-1} \sum_{k_1=0}^{N_1-1} -\left(k_1^2 + k_2^2\right) \left(\sum_{n_2=0}^{N_2-1} \sum_{n_1=0}^{N_1-1} u[n_1, n_2] \omega_{N_1}^{-n_1 k_1} \omega_{N_2}^{-n_2 k_2} \right) \omega_{N_1}^{n_1 k_1} \omega_{N_2}^{n_2 k_2}.$$

Classical approach

Use of state-of-the-art parallel FFTs routines, based on the transpose method
For the two dimensional (2D) case:

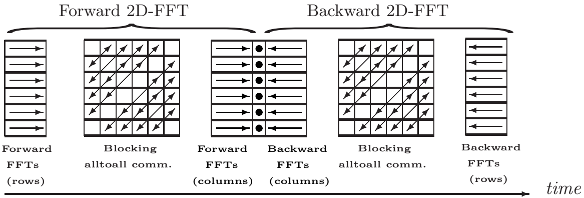
$$y[n_1, n_2] = \frac{1}{N} \sum_{k_2=0}^{N_2-1} \sum_{k_1=0}^{N_1-1} -\left(k_1^2 + k_2^2\right) \left(\sum_{n_2=0}^{N_2-1} \sum_{n_1=0}^{N_1-1} u[n_1, n_2] \omega_{N_1}^{-n_1 k_1} \omega_{N_2}^{-n_2 k_2} \right) \omega_{N_1}^{n_1 k_1} \omega_{N_2}^{n_2 k_2}.$$



Classical approach

Use of state-of-the-art parallel FFTs routines, based on the transpose method
For the two dimensional (2D) case:

$$y[n_1, n_2] = \frac{1}{N} \sum_{k_2=0}^{N_2-1} \sum_{k_1=0}^{N_1-1} -\left(k_1^2 + k_2^2\right) \left(\sum_{n_2=0}^{N_2-1} \sum_{n_1=0}^{N_1-1} u[n_1, n_2] \omega_{N_1}^{-n_1 k_1} \omega_{N_2}^{-n_2 k_2} \right) \omega_{N_1}^{n_1 k_1} \omega_{N_2}^{n_2 k_2}.$$

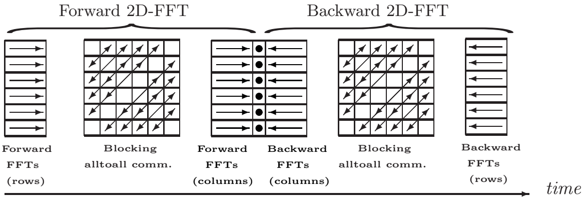


- Four computation steps separated by communications

Classical approach

Use of state-of-the-art parallel FFTs routines, based on the transpose method
For the two dimensional (2D) case:

$$y[n_1, n_2] = \frac{1}{N} \sum_{k_2=0}^{N_2-1} \sum_{k_1=0}^{N_1-1} -\left(k_1^2 + k_2^2\right) \left(\sum_{n_2=0}^{N_2-1} \sum_{n_1=0}^{N_1-1} u[n_1, n_2] \omega_{N_1}^{-n_1 k_1} \omega_{N_2}^{-n_2 k_2} \right) \omega_{N_1}^{n_1 k_1} \omega_{N_2}^{n_2 k_2}.$$

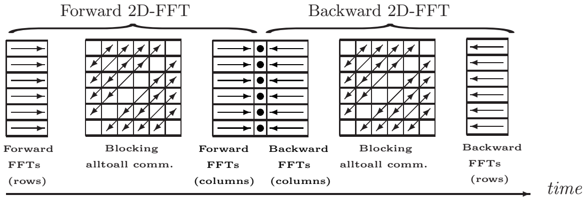


- Four computation steps separated by communications
- Dimensions are sequentially computed in a divide-and-conquer strategy

Classical approach

Use of state-of-the-art parallel FFTs routines, based on the transpose method
For the two dimensional (2D) case:

$$y[n_1,n_2] = \frac{1}{N}\sum_{k_2=0}^{N_2-1}\sum_{k_1=0}^{N_1-1} -\left(k_1^2+k_2^2\right)\left(\sum_{n_2=0}^{N_2-1}\sum_{n_1=0}^{N_1-1} u[n_1,n_2]\omega_{N_1}^{-n_1k_1}\omega_{N_2}^{-n_2k_2}\right)\omega_{N_1}^{n_1k_1}\omega_{N_2}^{n_2k_2}.$$



- Four computation steps separated by communications
- Dimensions are sequentially computed in a divide-and-conquer strategy
- Blocking communications assure such strict sequencing

The inherent overlapping in the Laplacian

$$y[n_1, n_2] = \frac{1}{N_1}\sum_{k_1=0}^{N_1-1}\left(-k_1^2\sum_{n_1=0}^{N_1-1}u[n_1, n_2]\omega_{N_1}^{-k_1 n_1}\right)\omega_{N_1}^{k_1 n_1} + \frac{1}{N_2}\sum_{k_2=0}^{N_2-1}\left(-k_2^2\sum_{n_2=0}^{N_2-1}u[n_1, n_2]\omega_{N_2}^{-k_2 n_2}\right)\omega_{N_2}^{k_2 n_2}$$

The inherent overlapping in the Laplacian

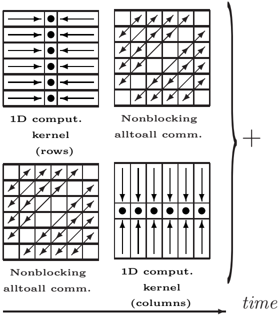
The linear property of the FFT turns a single 2D problem turns into two independent 1D problems

$$y[n_1, n_2] = \frac{1}{N_1}\sum_{k_1=0}^{N_1-1}\left(-k_1^2\sum_{n_1=0}^{N_1-1}u[n_1, n_2]\omega_{N_1}^{-k_1 n_1}\right)\omega_{N_1}^{k_1 n_1} + \frac{1}{N_2}\sum_{k_2=0}^{N_2-1}\left(-k_2^2\sum_{n_2=0}^{N_2-1}u[n_1, n_2]\omega_{N_2}^{-k_2 n_2}\right)\omega_{N_2}^{k_2 n_2}$$

The inherent overlapping in the Laplacian

The linear property of the FFT turns a single 2D problem turns into two independent 1D problems

$$y[n_1, n_2] = \frac{1}{N_1} \sum_{k_1=0}^{N_1-1} \left(-k_1^2 \sum_{n_1=0}^{N_1-1} u[n_1, n_2] \omega_{N_1}^{-k_1 n_1} \right) \omega_{N_1}^{k_1 n_1} + \frac{1}{N_2} \sum_{k_2=0}^{N_2-1} \left(-k_2^2 \sum_{n_2=0}^{N_2-1} u[n_1, n_2] \omega_{N_2}^{-k_2 n_2} \right) \omega_{N_2}^{k_2 n_2}$$

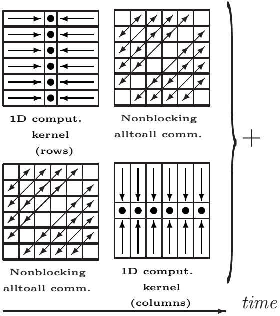


The inherent overlapping in the Laplacian

The linear property of the FFT turns a single 2D problem turns into two independent 1D problems

$$y[n_1,n_2] = \frac{1}{N_1}\sum_{k_1=0}^{N_1-1}\left(-k_1^2\sum_{n_1=0}^{N_1-1}u[n_1,n_2]\omega_{N_1}^{-k_1 n_1}\right)\omega_{N_1}^{k_1 n_1} + \frac{1}{N_2}\sum_{k_2=0}^{N_2-1}\left(-k_2^2\sum_{n_2=0}^{N_2-1}u[n_1,n_2]\omega_{N_2}^{-k_2 n_2}\right)\omega_{N_2}^{k_2 n_2}$$

- Left column: one kernel (rows) is computed while a copy of data is currently been sent

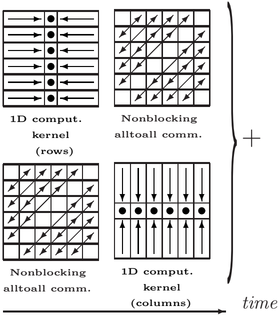


The inherent overlapping in the Laplacian

The linear property of the FFT turns a single 2D problem turns into two independent 1D problems

$$y[n_1, n_2] = \frac{1}{N_1} \sum_{k_1=0}^{N_1-1} \left(-k_1^2 \sum_{n_1=0}^{N_1-1} u[n_1, n_2] \omega_{N_1}^{-k_1 n_1} \right) \omega_{N_1}^{k_1 n_1} + \frac{1}{N_2} \sum_{k_2=0}^{N_2-1} \left(-k_2^2 \sum_{n_2=0}^{N_2-1} u[n_1, n_2] \omega_{N_2}^{-k_2 n_2} \right) \omega_{N_2}^{k_2 n_2}$$

- Left column: one kernel (rows) is computed while a copy of data is currently been sent
- Right column: the result on the rows is sent while data on columns are still being computed

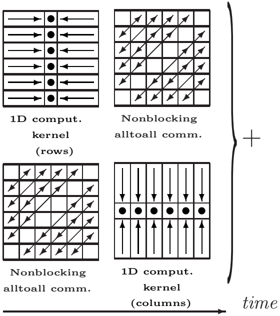


The inherent overlapping in the Laplacian

The linear property of the FFT turns a single 2D problem turns into two independent 1D problems

$$y[n_1,n_2] = \frac{1}{N_1}\sum_{k_1=0}^{N_1-1}\left(-k_1^2\sum_{n_1=0}^{N_1-1}u[n_1,n_2]\omega_{N_1}^{-k_1n_1}\right)\omega_{N_1}^{k_1n_1} + \frac{1}{N_2}\sum_{k_2=0}^{N_2-1}\left(-k_2^2\sum_{n_2=0}^{N_2-1}u[n_1,n_2]\omega_{N_2}^{-k_2n_2}\right)\omega_{N_2}^{k_2n_2}$$

- Left column: one kernel (rows) is computed while a copy of data is currently been sent
- Right column: the result on the rows is sent while data on columns are still being computed
- A final sum finishes the computation

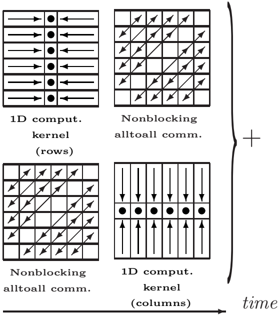


The inherent overlapping in the Laplacian

The linear property of the FFT turns a single 2D problem turns into two independent 1D problems

$$y[n_1, n_2] = \frac{1}{N_1}\sum_{k_1=0}^{N_1-1}\left(-k_1^2\sum_{n_1=0}^{N_1-1}u[n_1, n_2]\omega_{N_1}^{-k_1 n_1}\right)\omega_{N_1}^{k_1 n_1} + \frac{1}{N_2}\sum_{k_2=0}^{N_2-1}\left(-k_2^2\sum_{n_2=0}^{N_2-1}u[n_1, n_2]\omega_{N_2}^{-k_2 n_2}\right)\omega_{N_2}^{k_2 n_2}$$

- Left column: one kernel (rows) is computed while a copy of data is currently been sent
- Right column: the result on the rows is sent while data on columns are still being computed
- A final sum finishes the computation
- Same number of blocks: communication bandwidth and computational load FLOPS are the same

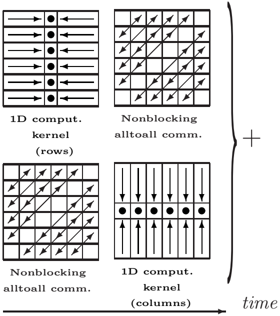


The inherent overlapping in the Laplacian

The linear property of the FFT turns a single 2D problem turns into two independent 1D problems

$$y[n_1,n_2] = \frac{1}{N_1}\sum_{k_1=0}^{N_1-1}\left(-k_1^2\sum_{n_1=0}^{N_1-1}u[n_1,n_2]\omega_{N_1}^{-k_1 n_1}\right)\omega_{N_1}^{k_1 n_1} + \frac{1}{N_2}\sum_{k_2=0}^{N_2-1}\left(-k_2^2\sum_{n_2=0}^{N_2-1}u[n_1,n_2]\omega_{N_2}^{-k_2 n_2}\right)\omega_{N_2}^{k_2 n_2}$$

- Left column: one kernel (rows) is computed while a copy of data is currently been sent
- Right column: the result on the rows is sent while data on columns are still being computed
- A final sum finishes the computation
- Same number of blocks: communication bandwidth and computational load FLOPS are the same
- The potential benefits come from overlapping



Performance tests and results

Performance tests and results

- Our approach implemented in `C`. Nonblocking communications rely on the `MPI_Ialltoall` function and local computations on the `FFTW`

Performance tests and results

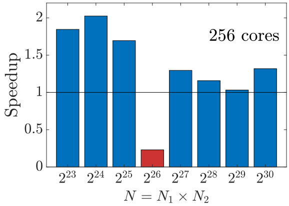
- Our approach implemented in `C`. Nonblocking communications rely on the `MPI_Ialltoall` function and local computations on the `FFTW`
- Classical solution based on the 2D FFT of the `MPI-FFTW` library

Performance tests and results

- Our approach implemented in `C`. Nonblocking communications rely on the `MPI_Ialltoall` function and local computations on the `FFTW`
- Classical solution based on the 2D FFT of the `MPI-FFTW` library
- Both solutions are compared on SCAYLE (*Supercomputación CAstilla Y LEón*), where each node has 2 Haswell (octa-core) processors

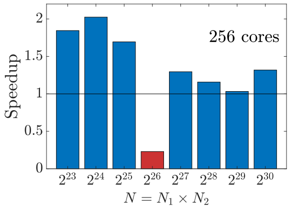
Performance tests and results

- Our approach implemented in `C`. Nonblocking communications rely on the `MPI_Ialltoall` function and local computations on the `FFTW`
- Classical solution based on the 2D FFT of the `MPI-FFTW` library
- Both solutions are compared on SCAYLE (*Supercomputación CAstilla Y LEón*), where each node has 2 Haswell (octa-core) processors
- Speedup figures for different problem sizes $N = N_1 \times N_2$ and cores P



Performance tests and results

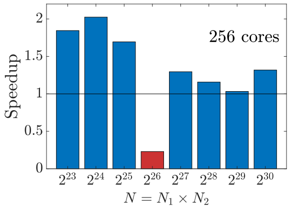
- Our approach implemented in `C`. Nonblocking communications rely on the `MPI_Ialltoall` function and local computations on the `FFTW`
- Classical solution based on the 2D FFT of the `MPI-FFTW` library
- Both solutions are compared on SCAYLE (*Supercomputación CAstilla Y LEón*), where each node has 2 Haswell (octa-core) processors
- Speedup figures for different problem sizes $N = N_1 \times N_2$ and cores P



- the overlapped solution provides a speedup (S) where $1 \leq S \leq 2$

Performance tests and results

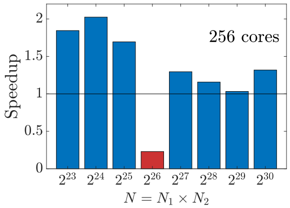
- Our approach implemented in C. Nonblocking communications rely on the MPI_Ialltoall function and local computations on the FFTW
- Classical solution based on the 2D FFT of the MPI-FFTW library
- Both solutions are compared on SCAYLE (*Supercomputación CAstilla Y LEón*), where each node has 2 Haswell (octa-core) processors
- Speedup figures for different problem sizes $N = N_1 \times N_2$ and cores P



- the overlapped solution provides a speedup (S) where $1 \leq S \leq 2$
- an exception always occurs at $N/P^2 = 2^{10}$ (communication issue)

Performance tests and results

- Our approach implemented in `C`. Nonblocking communications rely on the `MPI_Ialltoall` function and local computations on the `FFTW`
- Classical solution based on the 2D FFT of the `MPI-FFTW` library
- Both solutions are compared on SCAYLE (*Supercomputación CAstilla Y LEón*), where each node has 2 Haswell (octa-core) processors
- Speedup figures for different problem sizes $N = N_1 \times N_2$ and cores P



- the overlapped solution provides a speedup (S) where $1 \leq S \leq 2$
- an exception always occurs at $N/P^2 = 2^{10}$ (communication issue)
- the `FFTW-MPI` replicates the cache miss for $N/P^2 = 2^{11}$

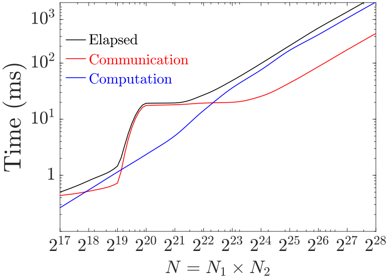
Discussion

Discussion

- Tests designed to answer whether overlapping is responsible for speedup

Discussion

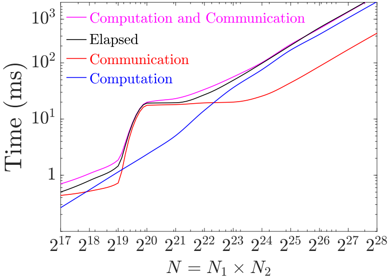
- Tests designed to answer whether overlapping is responsible for speedup
- **Elapsed**, computation and communication times are separately calculated for different problem sizes



- Computation evolves linearly since FLOPS are $\mathcal{O}(N \log N)$
- Communication undergoes the cache miss
- **Elapsed** is NOT the sum of both

Discussion

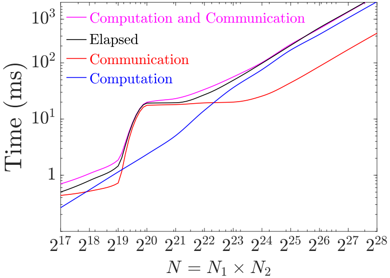
- Tests designed to answer whether overlapping is responsible for speedup
- **Elapsed**, computation and communication times are separately calculated for different problem sizes
- Sum of computation and communication has been added to the figure



- Computation evolves linearly since FLOPS are $\mathcal{O}(N \log N)$
- Communication undergoes the cache miss
- **Elapsed** is NOT the sum of both

Discussion

- Tests designed to answer whether overlapping is responsible for speedup
- **Elapsed**, computation and communication times are separately calculated for different problem sizes
- Sum of computation and communication has been added to the figure

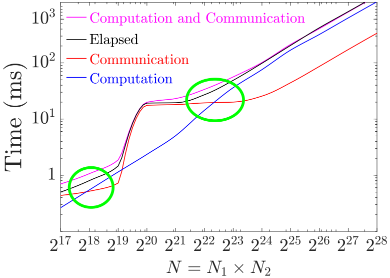


- Computation evolves linearly since FLOPS are $\mathcal{O}(N \log N)$
- Communication undergoes the cache miss
- **Elapsed** is NOT the sum of both

- Overlapping is represented by the difference between black and magenta

Discussion

- Tests designed to answer whether overlapping is responsible for speedup
- **Elapsed**, computation and communication times are separately calculated for different problem sizes
- Sum of computation and communication has been added to the figure

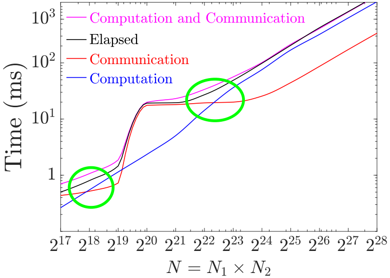


- Computation evolves linearly since FLOPS are $\mathcal{O}(N \log N)$
- Communication undergoes the cache miss
- **Elapsed** is NOT the sum of both

- Overlapping is represented by the difference between black and magenta
- The difference is maximum when the red and blue line cross, i.e. computation and computation times are comparable

Discussion

- Tests designed to answer whether overlapping is responsible for speedup
- **Elapsed**, computation and communication times are separately calculated for different problem sizes
- Sum of computation and communication has been added to the figure



- Computation evolves linearly since FLOPS are $\mathcal{O}(N \log N)$
- Communication undergoes the cache miss
- **Elapsed** is NOT the sum of both

- Overlapping is represented by the difference between black and magenta
- The difference is maximum when the red and blue line cross, i.e. computation and computation times are comparable
- However, no benefits are achieved when one prevails over the other

Conclusions

Conclusions

1. An overlapped implementation for the parallel computation of the Laplace has been presented

Conclusions

1. An overlapped implementation for the parallel computation of the Laplace has been presented
2. Inherent features of the operator are exploited leading to data independency

Conclusions

1. An overlapped implementation for the parallel computation of the Laplace has been presented
2. Inherent features of the operator are exploited leading to data independency
3. Computation and communications jobs are overlapped based on nonblocking communications

Conclusions

1. An overlapped implementation for the parallel computation of the Laplace has been presented
2. Inherent features of the operator are exploited leading to data independency
3. Computation and communications jobs are overlapped based on nonblocking communications
4. Our overlapped solution is compared to the conventional approach, based on parallel 2D FFTs

Conclusions

1. An overlapped implementation for the parallel computation of the Laplace has been presented
2. Inherent features of the operator are exploited leading to data independency
3. Computation and communications jobs are overlapped based on nonblocking communications
4. Our overlapped solution is compared to the conventional approach, based on parallel 2D FFTs
5. Speedup lies between 1 and 2 with one exception

Conclusions

1. An overlapped implementation for the parallel computation of the Laplace has been presented
2. Inherent features of the operator are exploited leading to data independency
3. Computation and communications jobs are overlapped based on nonblocking communications
4. Our overlapped solution is compared to the conventional approach, based on parallel 2D FFTs
5. Speedup lies between 1 and 2 with one exception
6. Overlapping is responsible for the good speedup figures

Conclusions

1. An overlapped implementation for the parallel computation of the Laplace has been presented
2. Inherent features of the operator are exploited leading to data independency
3. Computation and communications jobs are overlapped based on nonblocking communications
4. Our overlapped solution is compared to the conventional approach, based on parallel 2D FFTs
5. Speedup lies between 1 and 2 with one exception
6. Overlapping is responsible for the good speedup figures
7. Main programmer task: find this scenario (if possible) and work around it