Recent Advances in the Pseudospectral Method

John C. Bowman

University of Alberta

Dec 8, 2015

www.math.ualberta.ca/∼bowman/talks

Acknowledgements: Malcolm Roberts (Université de Strasbourg)

Pseudospectral Method

- Pseudospectral simulations are a widely used numerical tool for the study of fluid turbulence:
  - Fast $N \log N$ scaling for N modes.
  - Spectral accuracy: more accurate than finite-difference or finite-element methods.
- Ideal choice for studying homogenous turbulence with periodic boundary conditions.
- Generalizations such as Chebyshev collocation and penalty methods allow them to handle more complicated boundary conditions and geometries.
- However, in many cases pseudospectral methods do not parallelize well on massively parallel distributed architectures due to the communication costs of the parallel transpose.

Dealiasing

- Patterson and Orszag pioneered the pseudospectral method over 40 years ago.
- They emphasized that the convolution theorem necessitates *dealiasing* unwanted harmonics arising from the periodicity of the discrete Fourier transform.

Discrete Cyclic Convolution

- The FFT provides an efficient tool for computing the *discrete cyclic convolution*

$$\sum_{p=0}^{N-1} F_p G_{k-p},$$

where the vectors F and G have period N.

- Define the *Nth primitive root of unity:*

$$\zeta_N = \exp\left(\frac{2\pi i}{N}\right).$$

- The fast Fourier transform method exploits the properties that $\zeta_N^r = \zeta_{N/r}$ and $\zeta_N^N = 1$.

- However, the pseudospectral method requires a *linear convolution.*

- The unnormalized *backwards discrete Fourier transform* of $\{F_k : k = 0, \ldots, N\}$ is

$$f_j \doteq \sum_{k=0}^{N-1} \zeta_N^{jk} F_k \qquad j = 0, \ldots, N-1.$$

- The corresponding *forward transform is*

$$F_k \doteq \frac{1}{N} \sum_{j=0}^{N-1} \zeta_N^{-kj} f_j \qquad k = 0, \ldots, N-1.$$

- The orthogonality of this transform pair follows from

$$\sum_{j=0}^{N-1} \zeta_N^{\ell j} = \begin{cases} N & \text{if } \ell = sN \text{ for } s \in \mathbb{Z}, \\ \dfrac{1 - \zeta_N^{\ell N}}{1 - \zeta_N^{\ell}} = 0 & \text{otherwise.} \end{cases}$$

Convolution Theorem

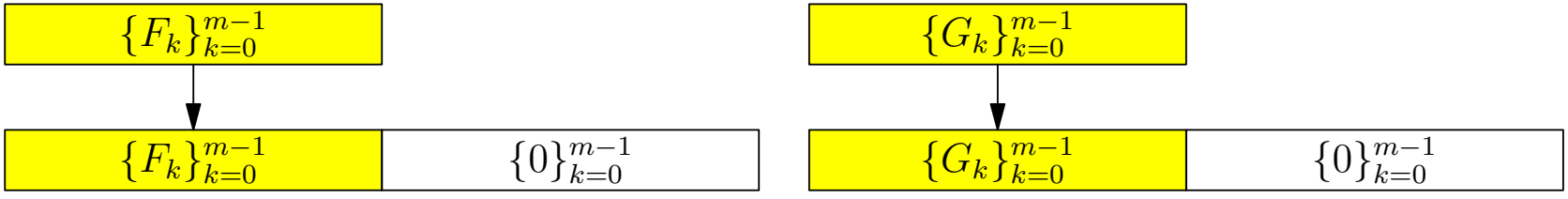
$$\begin{aligned}\sum_{j=0}^{N-1} f_j g_j \zeta_N^{-jk} &= \sum_{j=0}^{N-1} \zeta_N^{-jk} \left(\sum_{p=0}^{N-1} \zeta_N^{jp} F_p \right) \left(\sum_{q=0}^{N-1} \zeta_N^{jq} G_q \right) \\ &= \sum_{p=0}^{N-1} \sum_{q=0}^{N-1} F_p G_q \sum_{j=0}^{N-1} \zeta_N^{(-k+p+q)j} \\ &= N \sum_{s} \sum_{p=0}^{N-1} F_p G_{k-p+sN}.\end{aligned}$$

- The terms indexed by $s \neq 0$ are *aliases;* we need to remove them!
- If only the first m entries of the input vectors are nonzero, aliases can be avoided by *zero padding* input data vectors of length m to length $N \geq 2m - 1$.
- *Explicit zero padding* prevents mode $m - 1$ from beating with itself and wrapping around to contaminate mode $N = 0 \bmod N$.

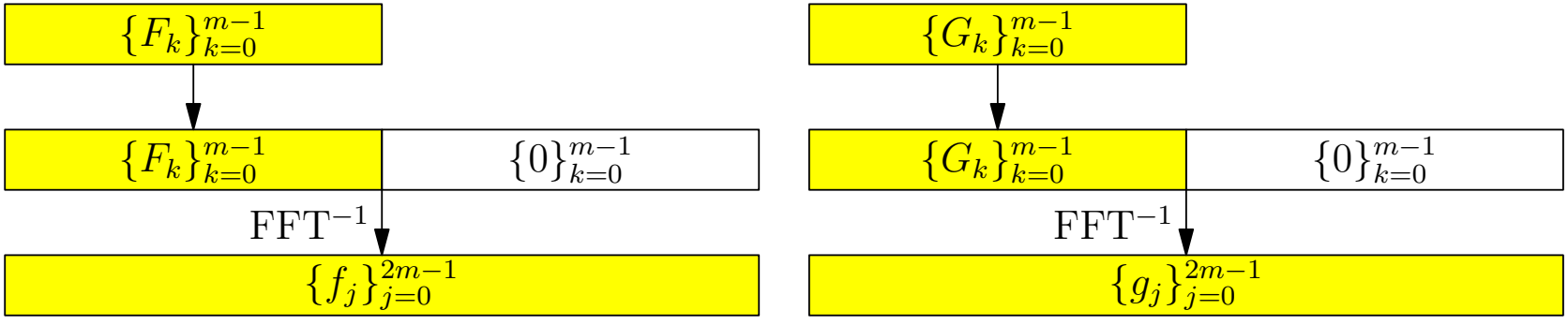
- Since FFT sizes with small prime factors in practice yield the most efficient implementations, the padding is normally extended to $N = 2m$:

$\{F_k\}_{k=0}^{m-1}$ $\{G_k\}_{k=0}^{m-1}$

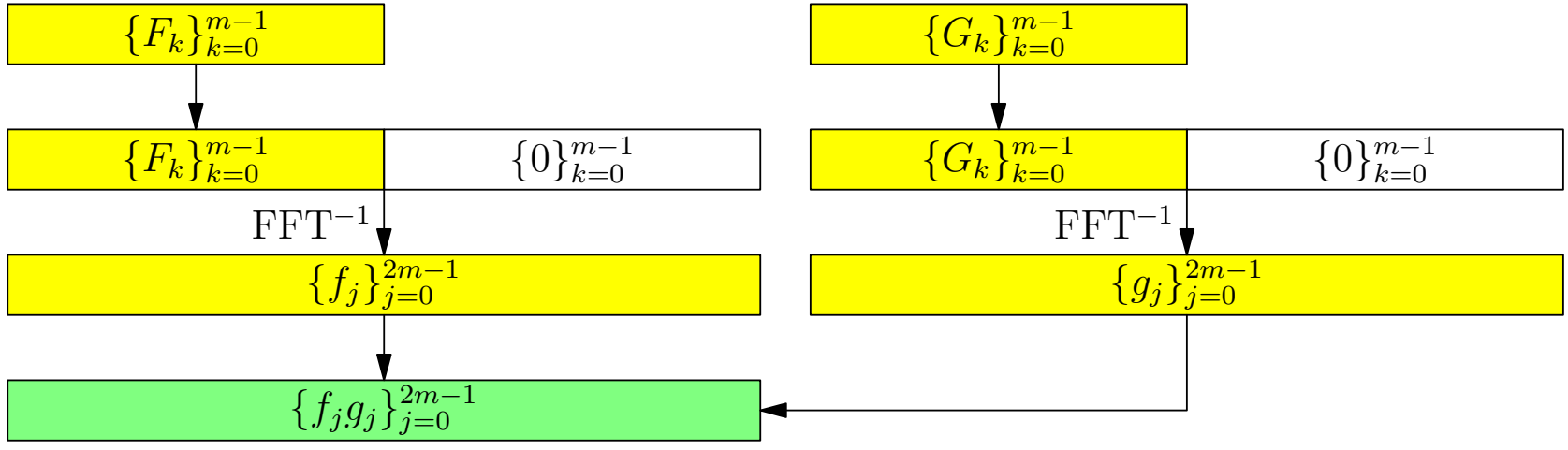
- Since FFT sizes with small prime factors in practice yield the most efficient implementations, the padding is normally extended to $N = 2m$:



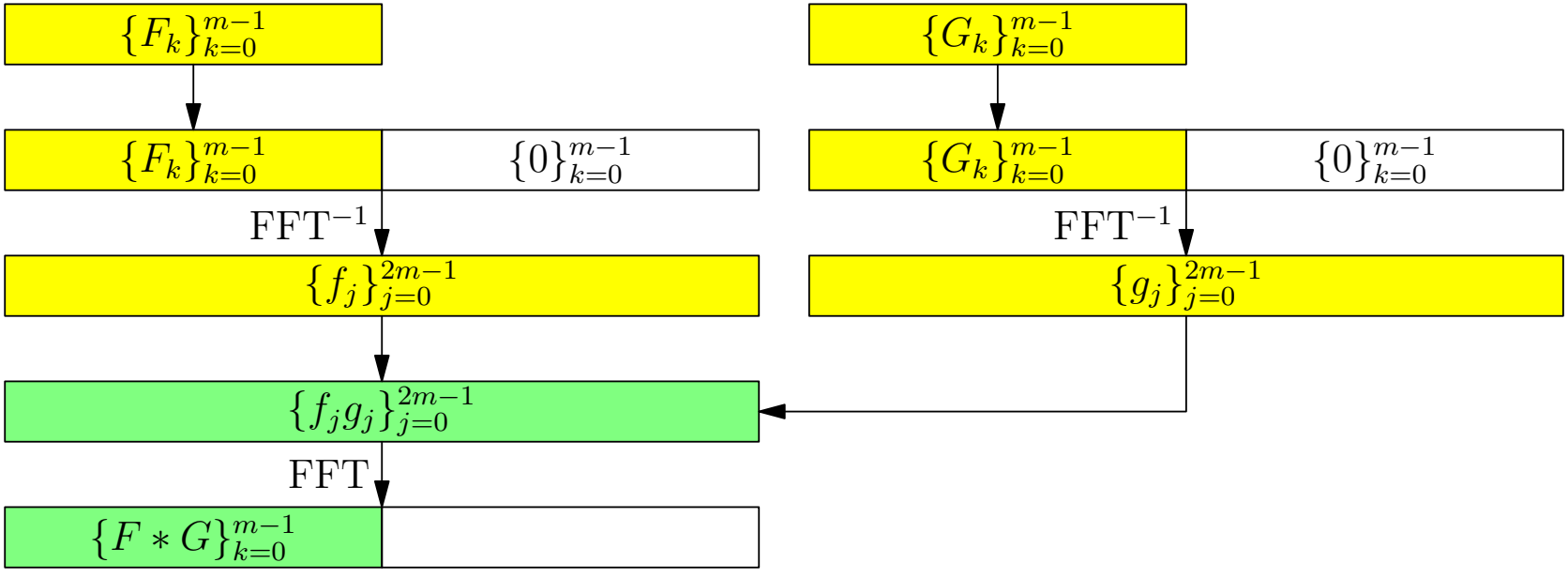
- Since FFT sizes with small prime factors in practice yield the most efficient implementations, the padding is normally extended to $N = 2m$:



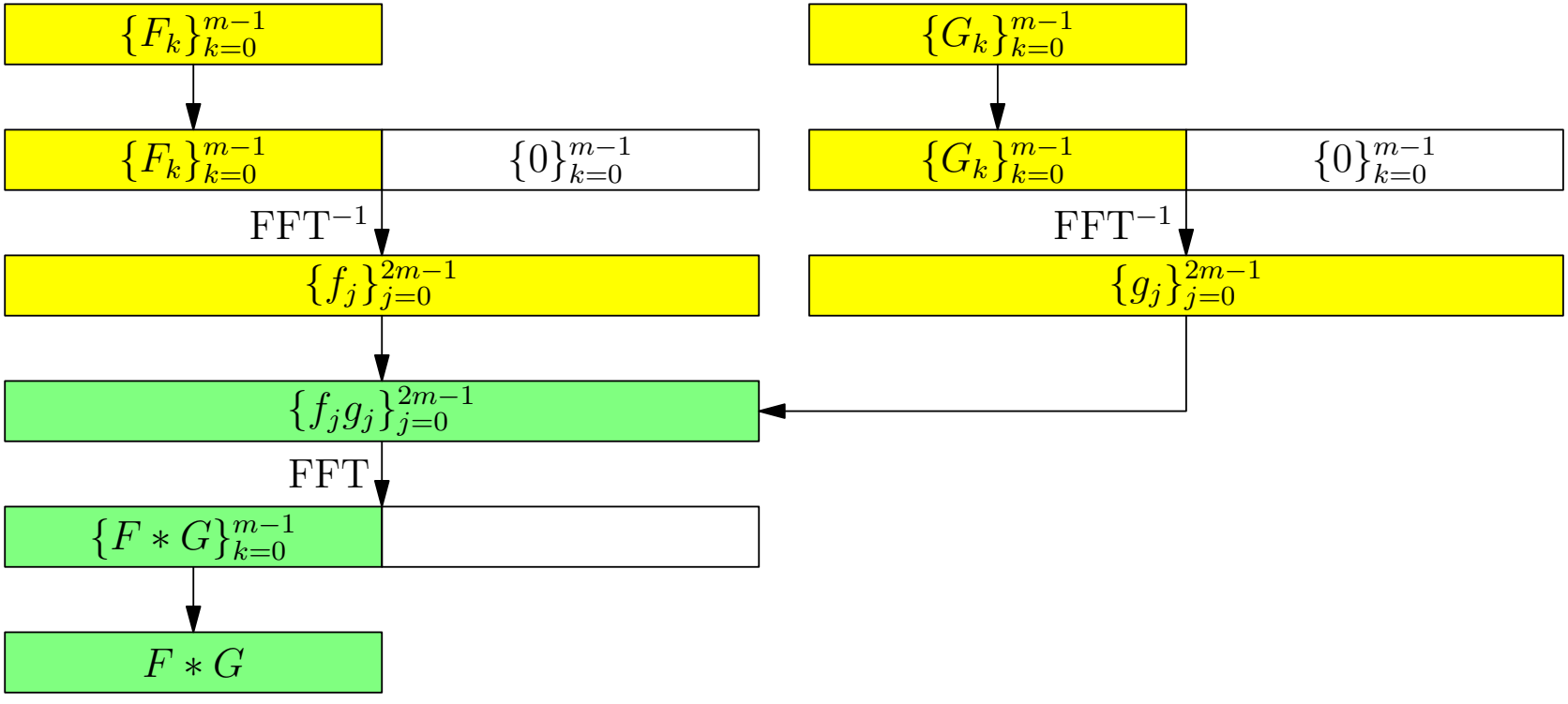
- Since FFT sizes with small prime factors in practice yield the most efficient implementations, the padding is normally extended to $N = 2m$:



- Since FFT sizes with small prime factors in practice yield the most efficient implementations, the padding is normally extended to $N = 2m$:



- Since FFT sizes with small prime factors in practice yield the most efficient implementations, the padding is normally extended to $N = 2m$:



Implicit Padding

- Let $N = 2m$. For $j = 0, \ldots, 2m-1$ we want to compute

$$f_j = \sum_{k=0}^{2m-1} \zeta_{2m}^{jk} F_k.$$

- If $F_k = 0$ for $k \geq m$, one can easily avoid looping over the unwanted zero Fourier modes by decimating in wavenumber:

$$f_{2\ell} = \sum_{k=0}^{m-1} \zeta_{2m}^{2\ell k} F_k = \sum_{k=0}^{m-1} \zeta_m^{\ell k} F_k,$$

$$f_{2\ell+1} = \sum_{k=0}^{m-1} \zeta_{2m}^{(2\ell+1)k} F_k = \sum_{k=0}^{m-1} \zeta_m^{\ell k} \zeta_{2m}^{k} F_k, \qquad \ell = 0, 1, \ldots m-1.$$

- This requires computing two subtransforms, each of size m, for an overall computational scaling of order $2m \log_2 m = N \log_2 m$.

- Odd and even terms of the convolution can then be computed separately, multiplied term-by-term, and transformed again to Fourier space:

$$
\begin{aligned}
2mF_k &= \sum_{j=0}^{2m-1} \zeta_{2m}^{-kj} f_j \\
&= \sum_{\ell=0}^{m-1} \zeta_{2m}^{-k2\ell} f_{2\ell} + \sum_{\ell=0}^{m-1} \zeta_{2m}^{-k(2\ell+1)} f_{2\ell+1} \\
&= \sum_{\ell=0}^{m-1} \zeta_{m}^{-k\ell} f_{2\ell} + \zeta_{2m}^{-k} \sum_{\ell=0}^{m-1} \zeta_{m}^{-k\ell} f_{2\ell+1} \qquad k = 0, \ldots, m-1.
\end{aligned}
$$

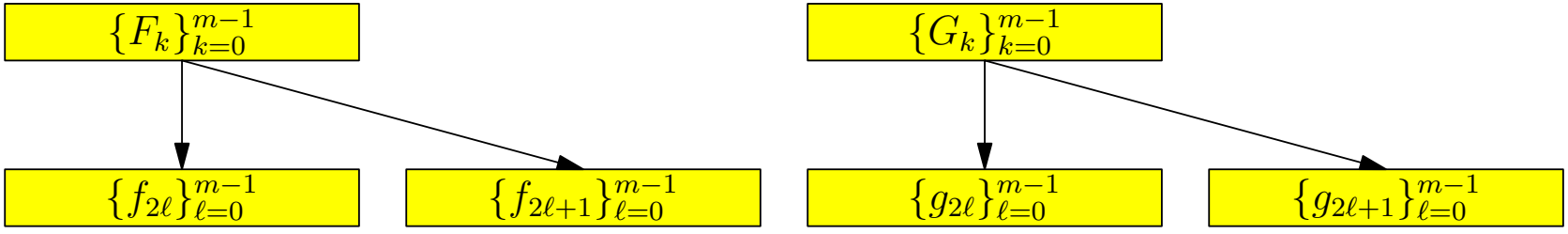
- No bit reversal is required at the highest level.
- A 1D implicitly padded convolution is implemented in our `FFTW++` library.
- This in-place convolution was written to use six out-of-place transforms, thereby avoiding bit reversal at all levels.

- The computational complexity is $6Km\log_2 m$.
- The numerical error is similar to explicit padding and the memory usage is identical.

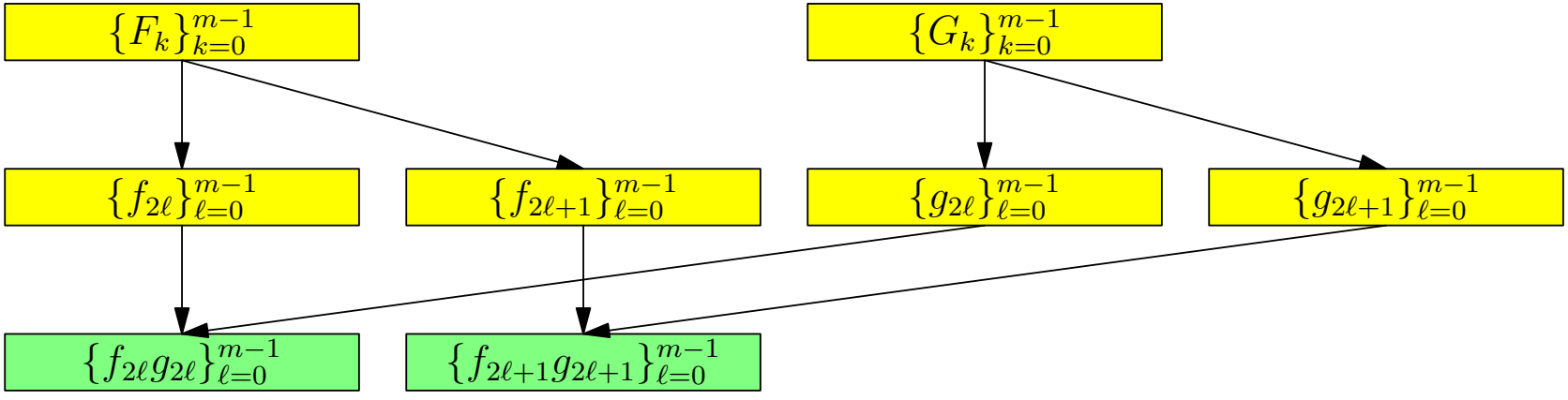
$\{F_k\}_{k=0}^{m-1}$

$\{G_k\}_{k=0}^{m-1}$

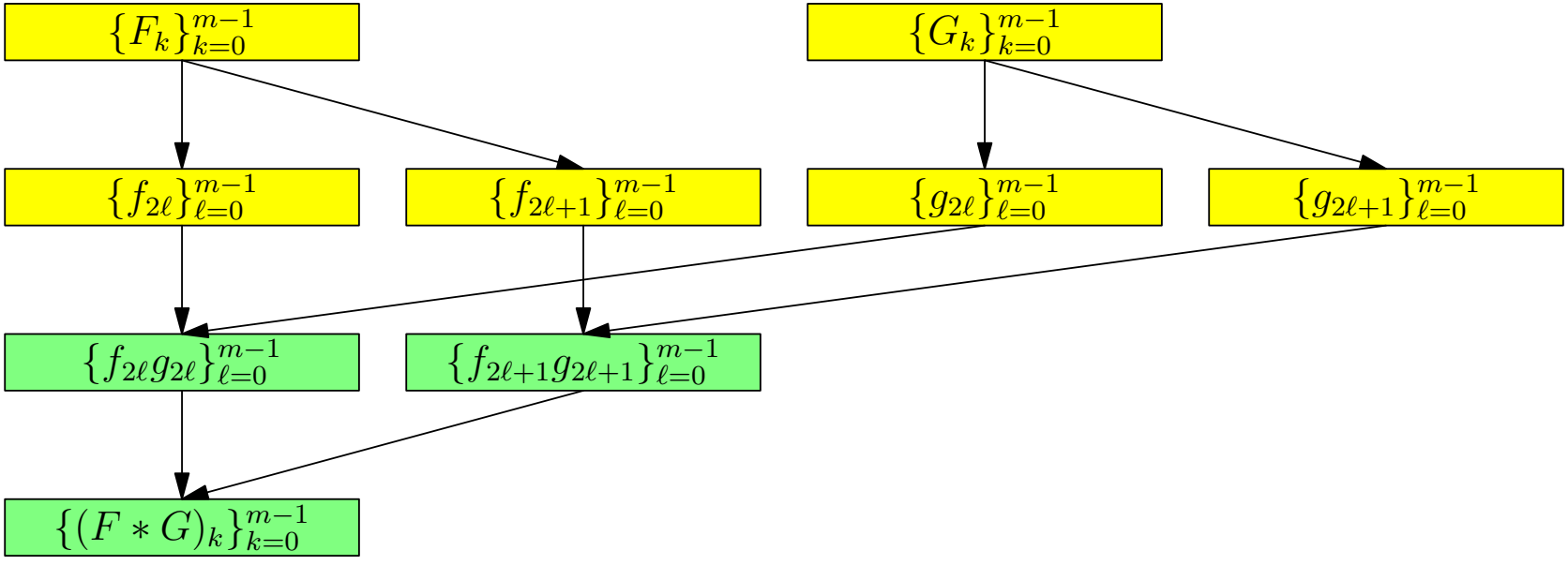
- The computational complexity is $6Km\log_2 m$.
- The numerical error is similar to explicit padding and the memory usage is identical.



- The computational complexity is $6Km\log_2 m$.
- The numerical error is similar to explicit padding and the memory usage is identical.

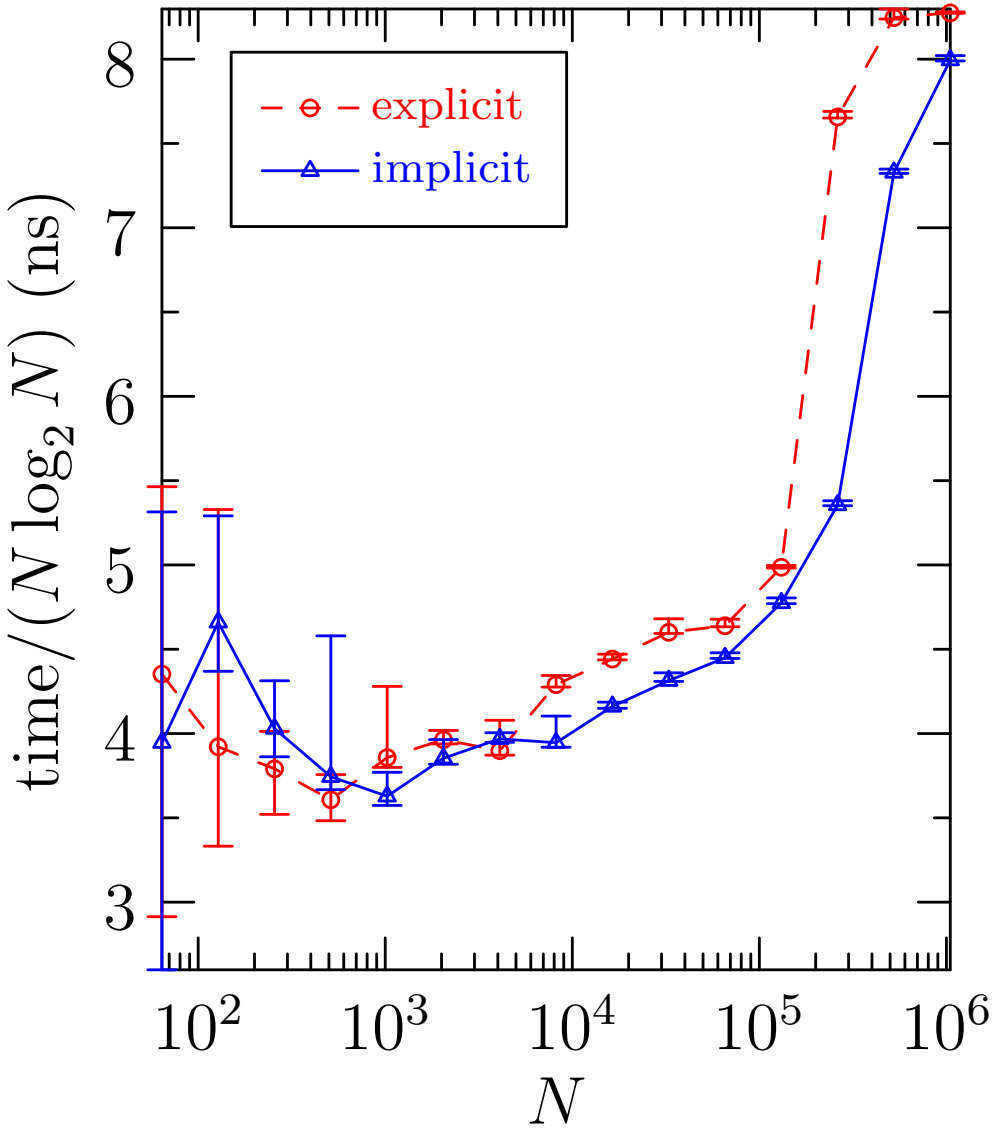


- The computational complexity is $6Km\log_2 m$.
- The numerical error is similar to explicit padding and the memory usage is identical.



Input: vector f, vector g
Output: vector f
$\mathsf{u} \leftarrow \mathtt{fft}^{-1}(\mathsf{f})$;
$\mathsf{v} \leftarrow \mathtt{fft}^{-1}(\mathsf{g})$;
$\mathsf{u} \leftarrow \mathsf{u} * \mathsf{v}$;
for $k = 0$ **to** $m - 1$ **do**
  $\mathsf{f}[k] \leftarrow \zeta_{2m}^{k} \mathsf{f}[k]$;
  $\mathsf{g}[k] \leftarrow \zeta_{2m}^{k} \mathsf{g}[k]$;
end
$\mathsf{v} \leftarrow \mathtt{fft}^{-1}(\mathsf{f})$;
$\mathsf{f} \leftarrow \mathtt{fft}^{-1}(\mathsf{g})$;
$\mathsf{v} \leftarrow \mathsf{v} * \mathsf{f}$;
$\mathsf{f} \leftarrow \mathtt{fft}(\mathsf{u})$;
$\mathsf{u} \leftarrow \mathtt{fft}(\mathsf{v})$;
for $k = 0$ **to** $m - 1$ **do**
  $\mathsf{f}[k] \leftarrow \mathsf{f}[k] + \zeta_{2m}^{-k} \mathsf{u}[k]$;
end
return f/(2m);

Implicit Padding in 1D



Convolutions in Higher Dimensions

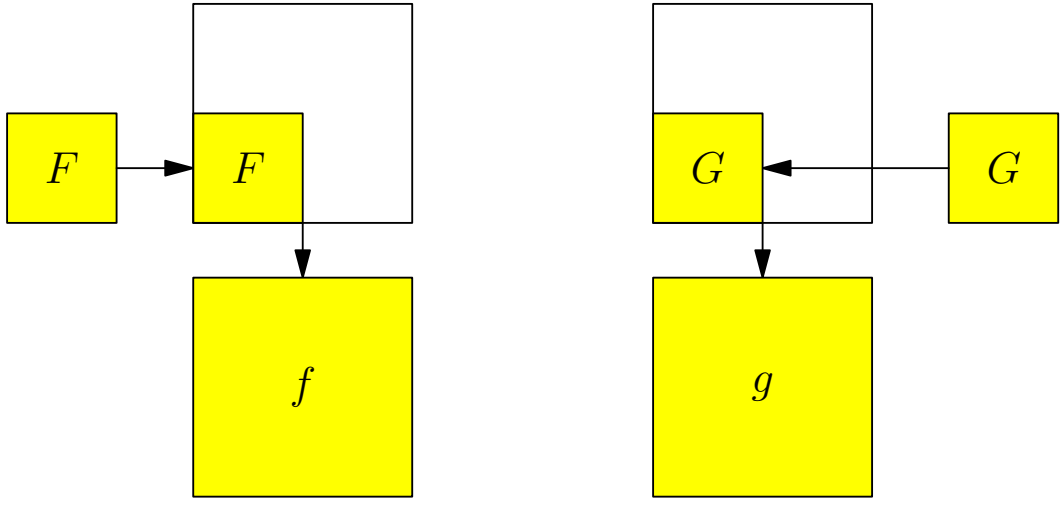
- An explicitly padded convolution in 2 dimensions requires 12 padded FFTs, and 4 times the memory of a cyclic convolution.

Convolutions in Higher Dimensions

- An explicitly padded convolution in 2 dimensions requires 12 padded FFTs, and 4 times the memory of a cyclic convolution.

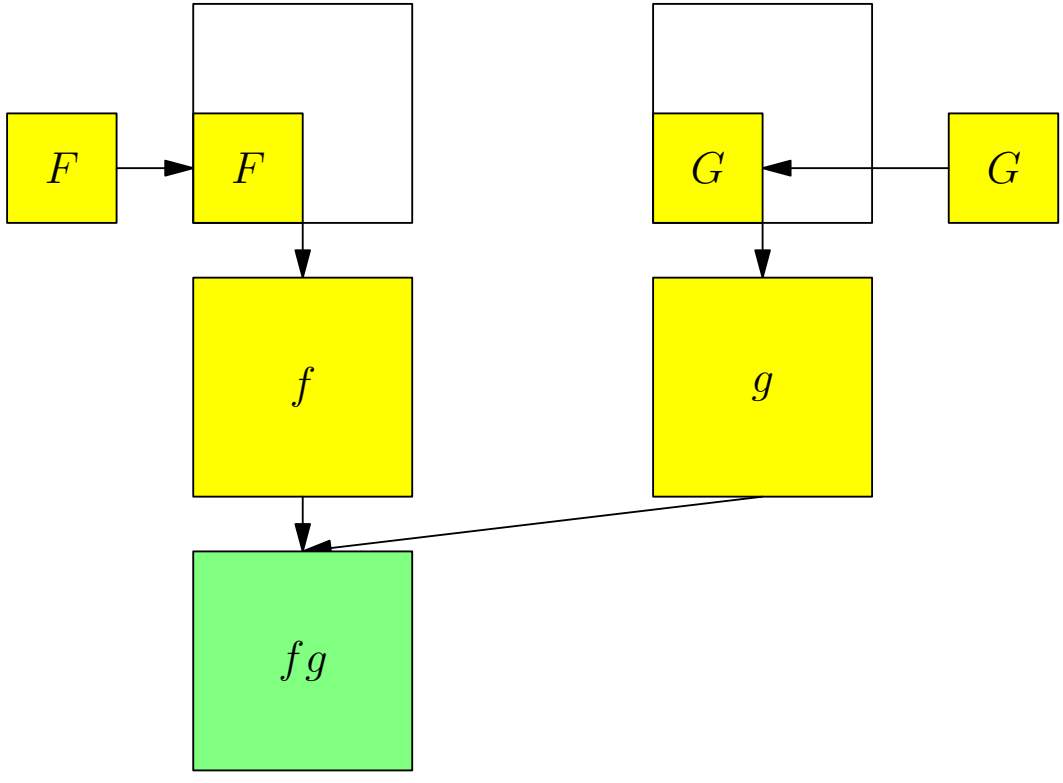
Convolutions in Higher Dimensions

- An explicitly padded convolution in 2 dimensions requires 12 padded FFTs, and 4 times the memory of a cyclic convolution.



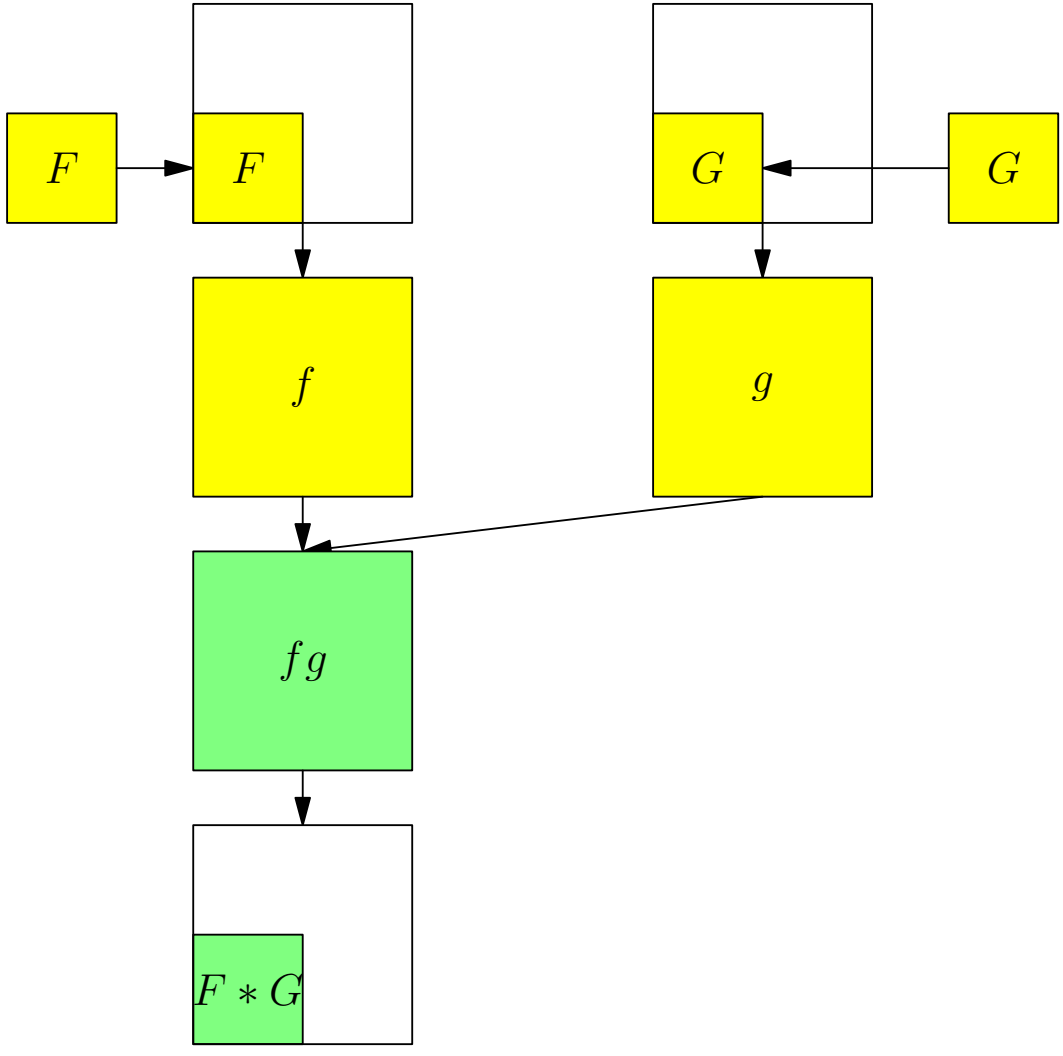
Convolutions in Higher Dimensions

- An explicitly padded convolution in 2 dimensions requires 12 padded FFTs, and 4 times the memory of a cyclic convolution.



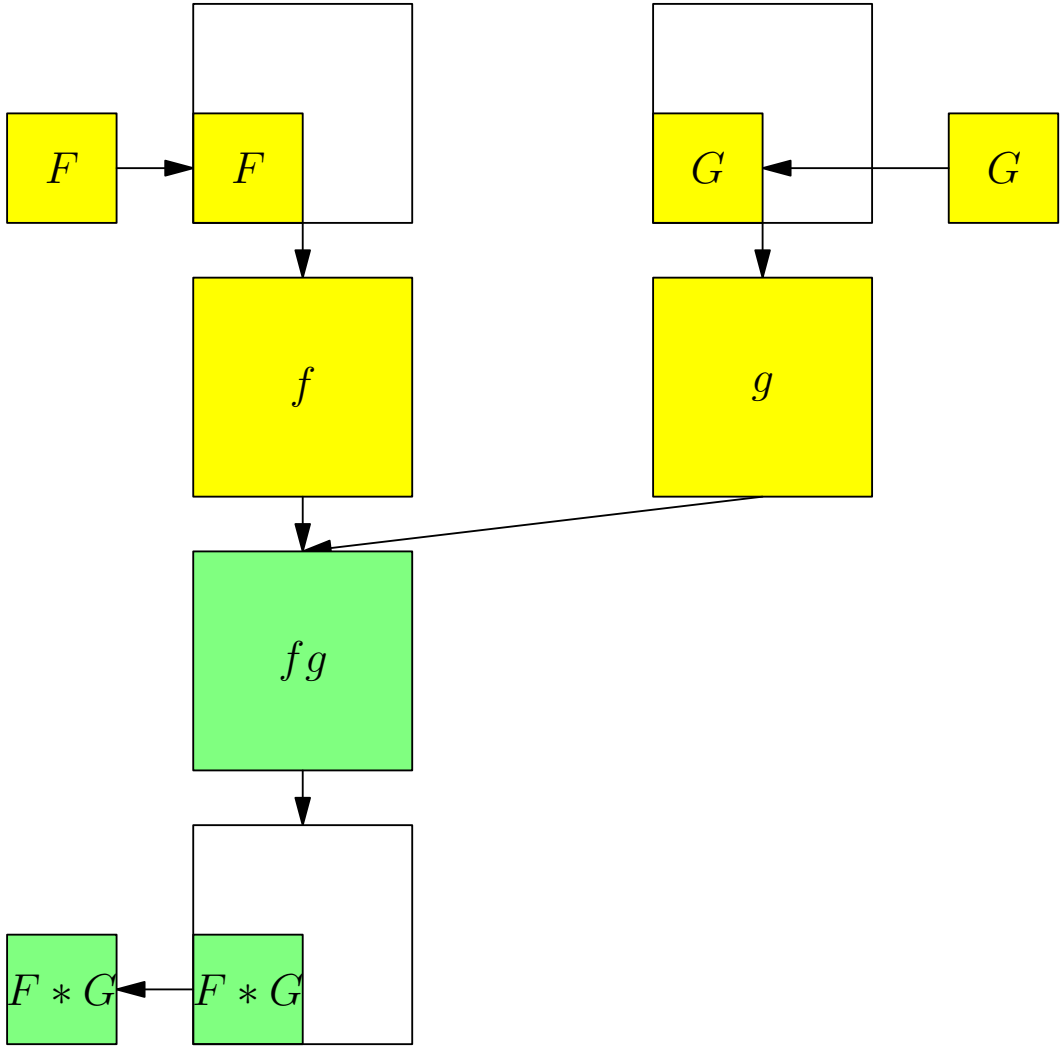
Convolutions in Higher Dimensions

- An explicitly padded convolution in 2 dimensions requires 12 padded FFTs, and 4 times the memory of a cyclic convolution.



Convolutions in Higher Dimensions

- An explicitly padded convolution in 2 dimensions requires 12 padded FFTs, and 4 times the memory of a cyclic convolution.



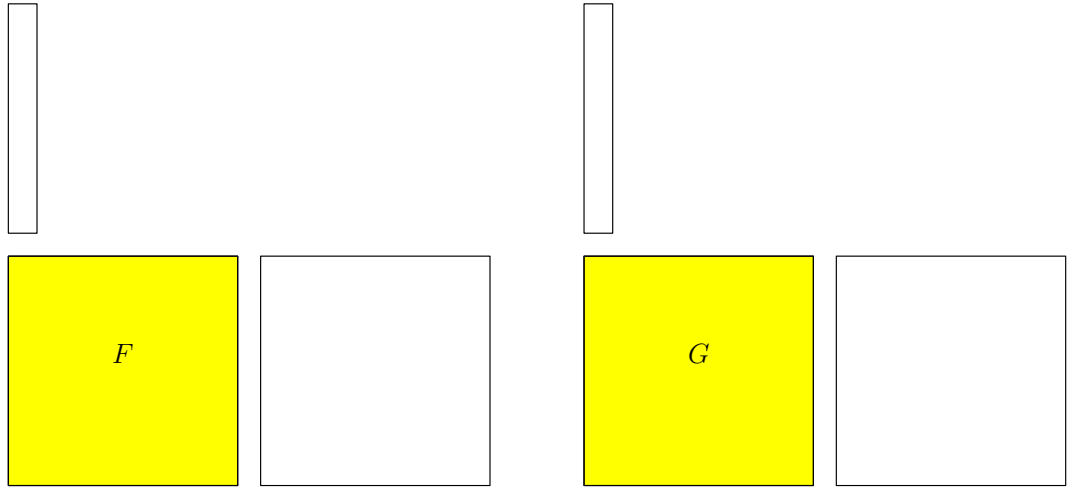
Recursive Convolution

- Naive way to compute a multiple-dimensional convolution:

- The technique of *recursive convolution* allows one to avoid computing and storing the entire Fourier image of the data:

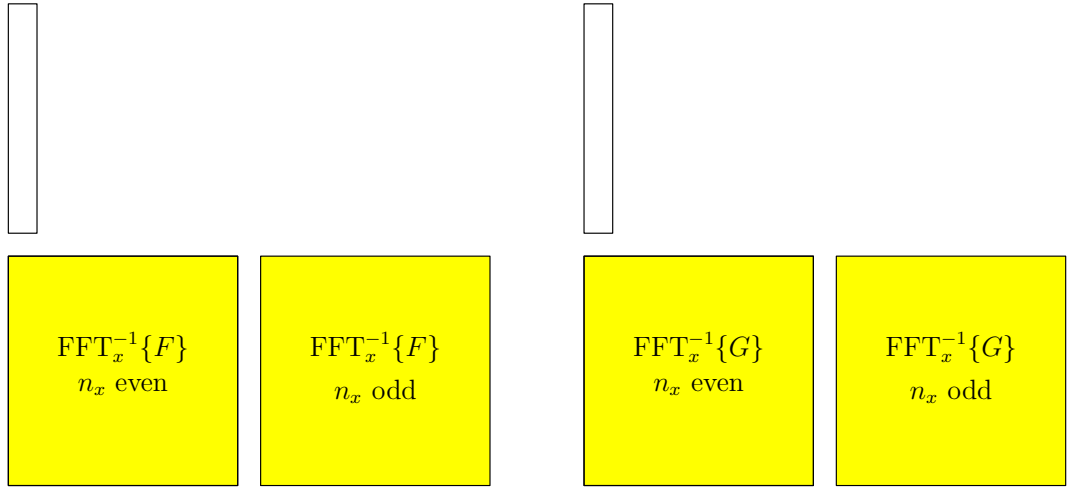
Implicit Padding in 2D

- Extra work memory need not be contiguous with the data.



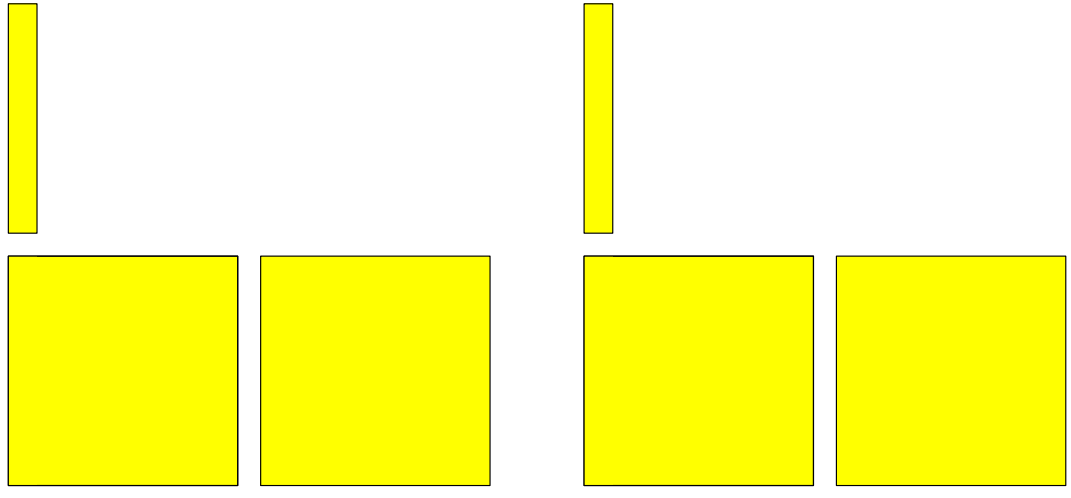
Implicit Padding in 2D

- Extra work memory need not be contiguous with the data.



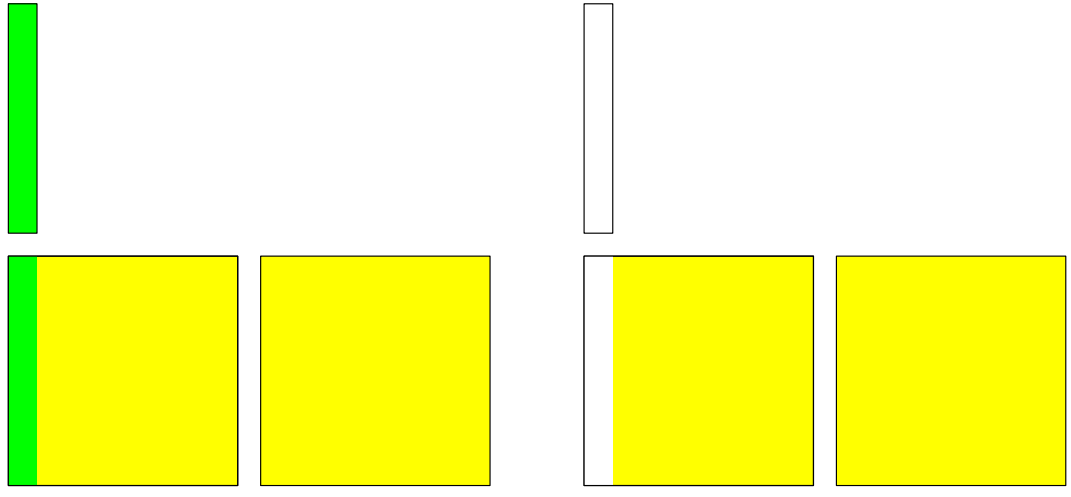
Implicit Padding in 2D

- Extra work memory need not be contiguous with the data.



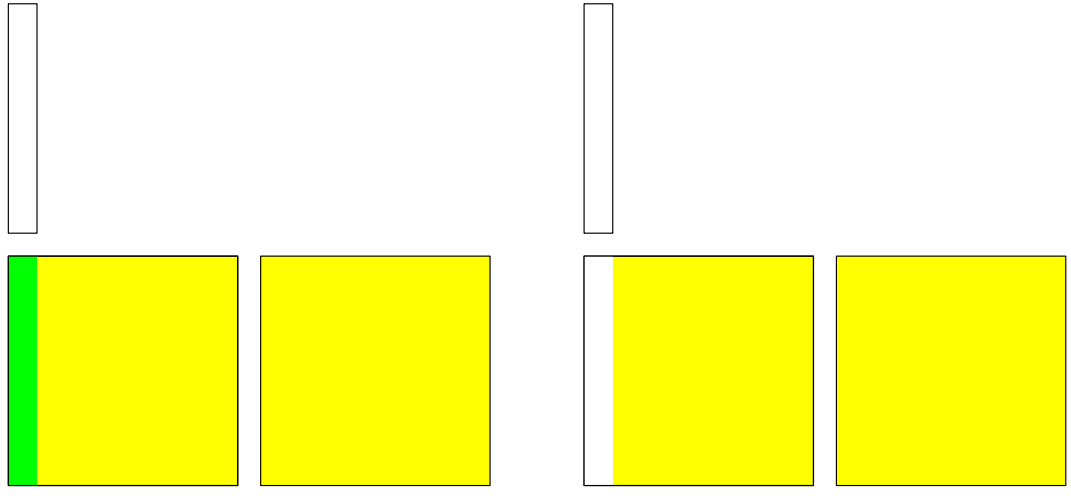
Implicit Padding in 2D

- Extra work memory need not be contiguous with the data.



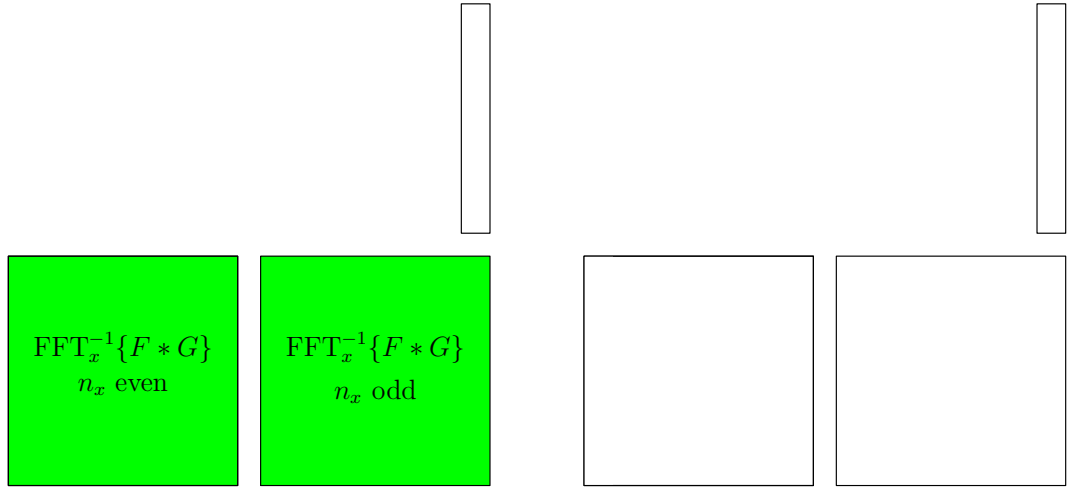
Implicit Padding in 2D

- Extra work memory need not be contiguous with the data.



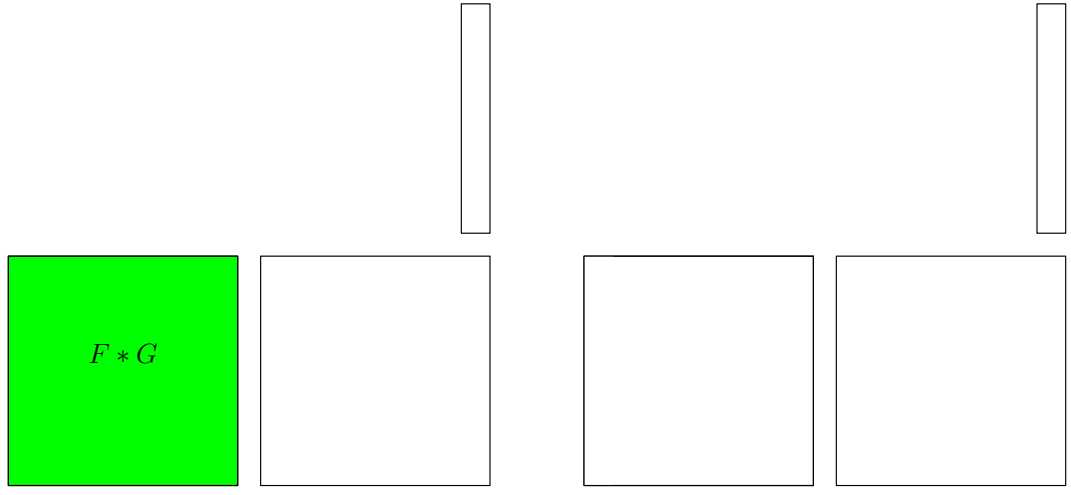
Implicit Padding in 2D

- Extra work memory need not be contiguous with the data.

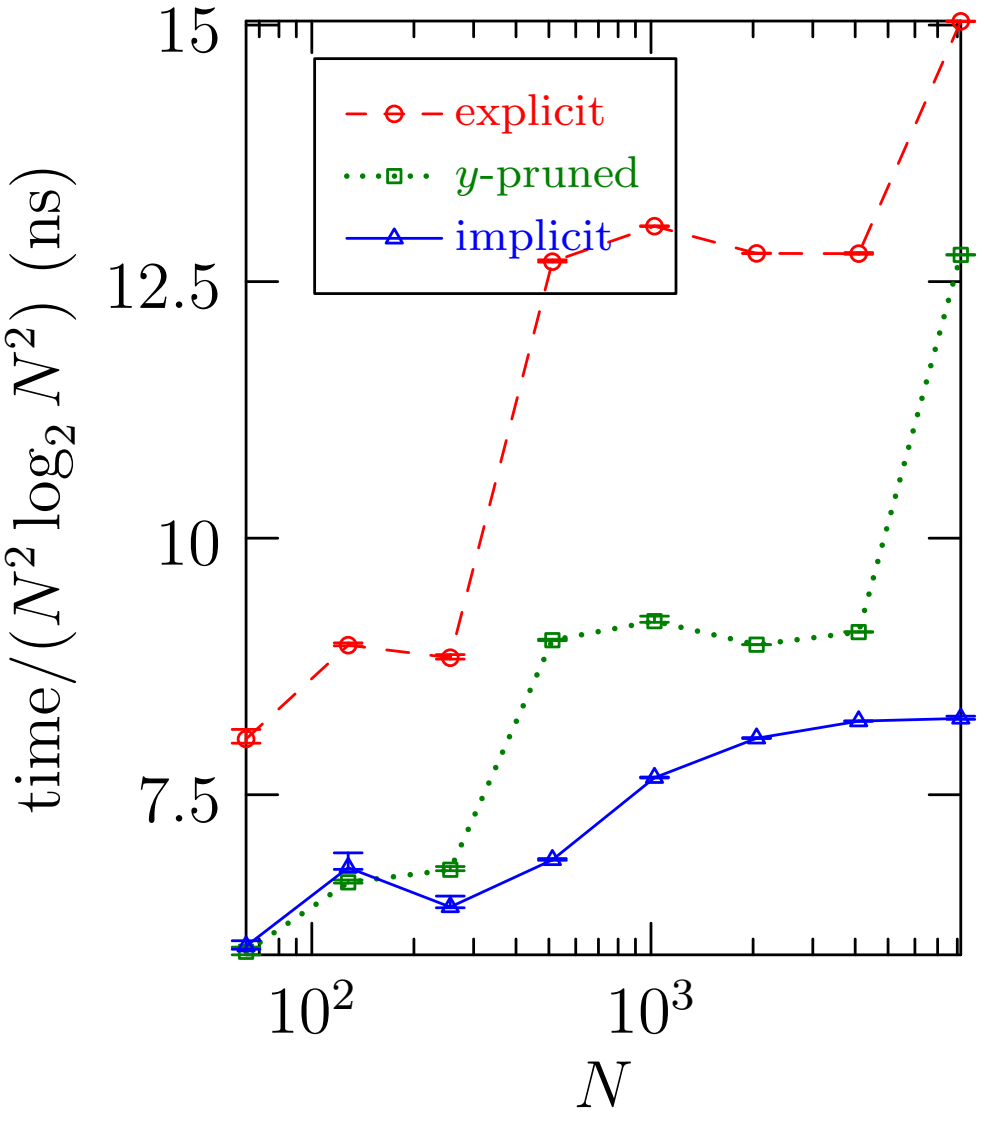


Implicit Padding in 2D

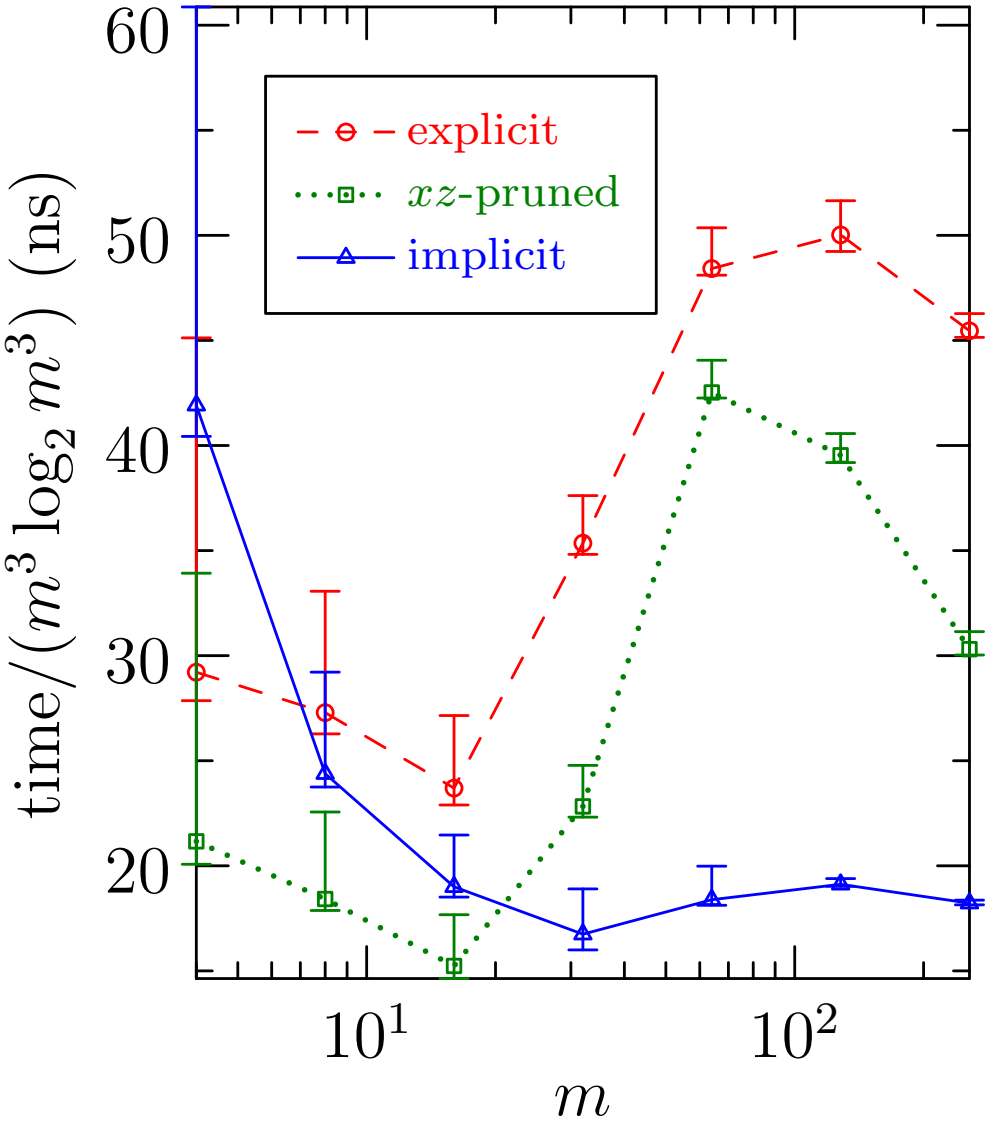
- Extra work memory need not be contiguous with the data.



Implicit Padding in 2D



Implicit Padding in 3D



Centered (Pseudospectral) Convolutions

- For a *centered convolution,* the Fourier origin ($k = 0$) is centered in the domain:

$$\sum_{p=k-m+1}^{m-1} f_p g_{k-p}$$

- Need to pad to $N \geq 3m - 2$ to remove aliases.
- The ratio $(2m-1)/(3m-2)$ of the number of physical to total modes is asymptotic to $2/3$ for large m.
- A *Hermitian convolution* arises since the input vectors are real:

$$f_{-k} = \overline{f_k}.$$

Multithreaded Hermitian Convolution

- The backwards implicitly padded centered Hermitian transform appears as

$$u_{3\ell+r} = \sum_{k=0}^{m-1} \zeta_m^{\ell k} w_{k,r},$$

where

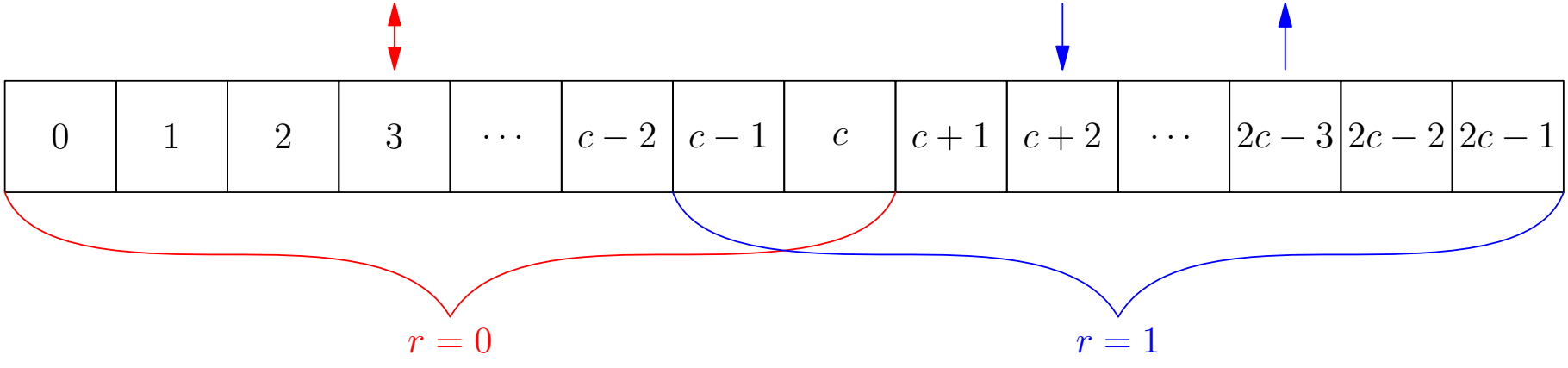
$$w_{k,r} \doteq \begin{cases} U_0 + \operatorname{Re} \zeta_3^{-r} U_{-m} & \text{if } k = 0, \\ \zeta_{3m}^{rk} \left(U_k + \zeta_3^{-r} \overline{U_{m-k}} \right) & \text{if } 1 \le k \le m-1. \end{cases}$$

- We exploit the Hermitian symmetry $w_{k,r} = \overline{w_{m-k,r}}$ to reduce the problem to three complex-to-real Fourier transforms of the first $c+1$ components of $w_{k,r}$ (one for each $r = -1, 0, 1$), where $c \doteq \lfloor m/2 \rfloor$ zeros.

- To facilitate an in-place implementation, in our original paper [SIAM J. Sci. Comput. 33, 386 (2011)], we stored the transformed values for $r = 1$ in reverse order in the upper half of the input vector.
- However, loop dependencies in the resulting algorithm prevented the top level of the 1D transforms from being multithreaded.

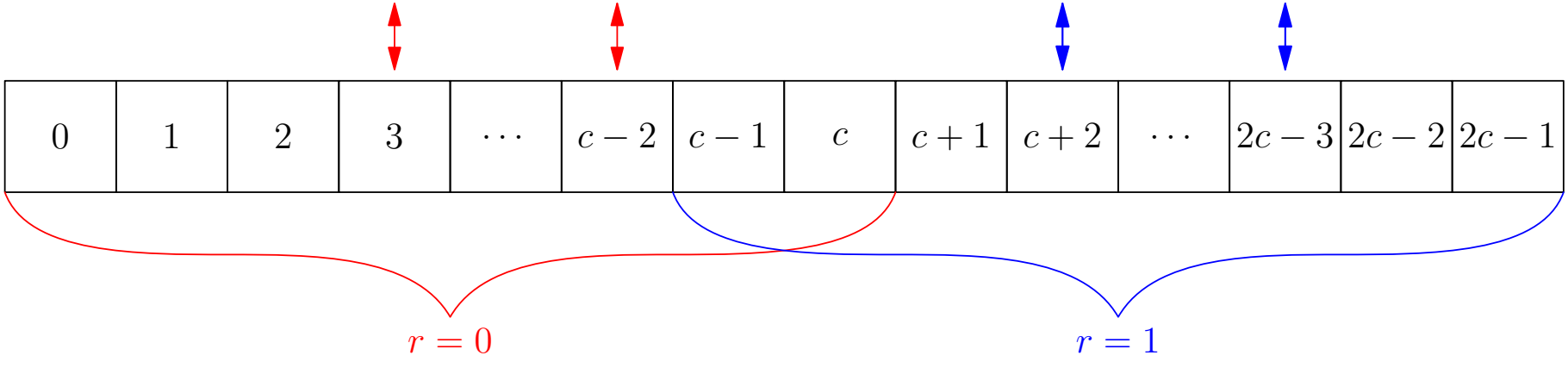
Multithreaded Hermitian Convolution

- Unrolling the loop to process four inputs and outputs simultaneously allows loop independence to be achieved, significantly improving performance in both the serial and parallel contexts.



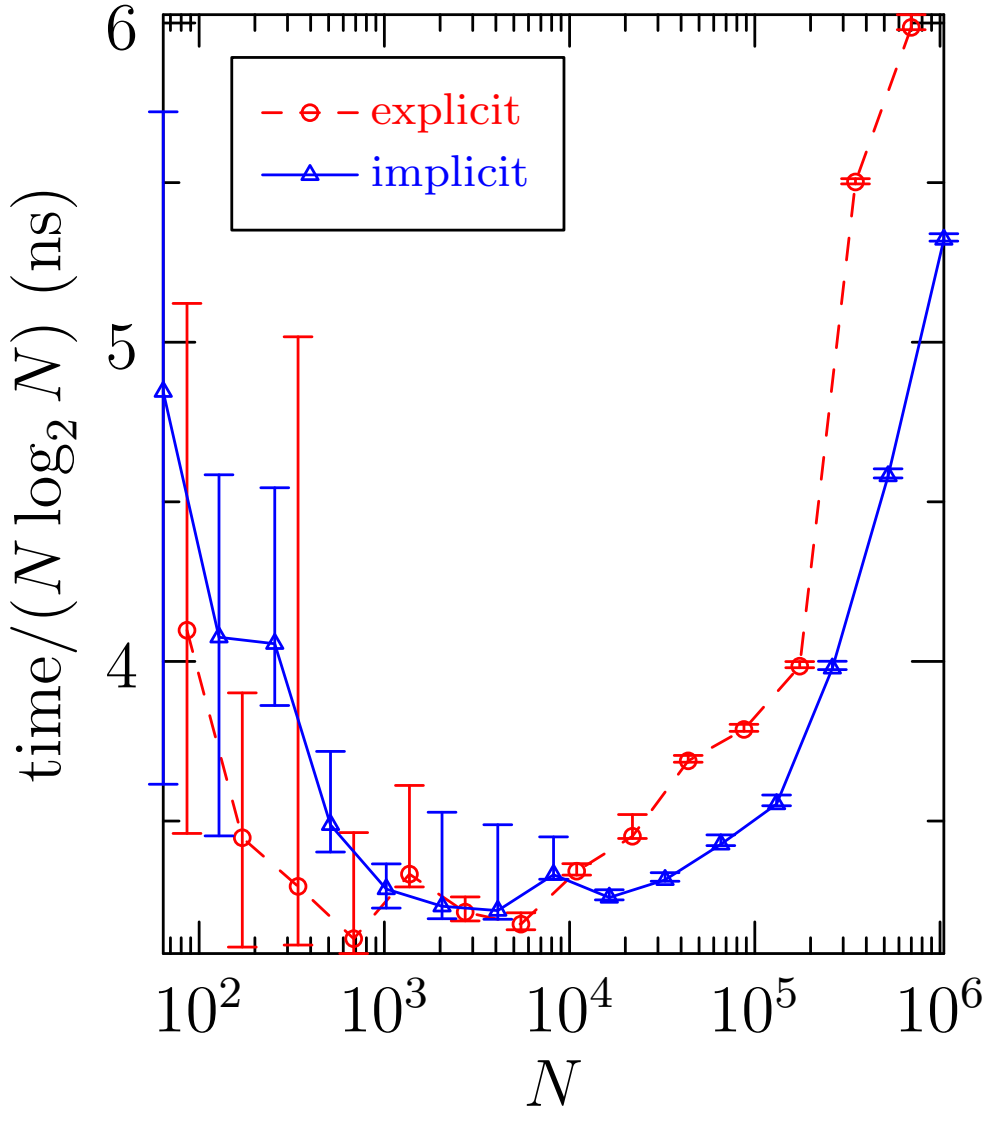
Multithreaded Hermitian Convolution

- Unrolling the loop to process four inputs and outputs simultaneously allows loop independence to be achieved, significantly improving performance in both the serial and parallel contexts.



- As a result, even in 1D, implicit dealiasing of pseudospectral convolutions is now significantly faster than explicit zero padding.

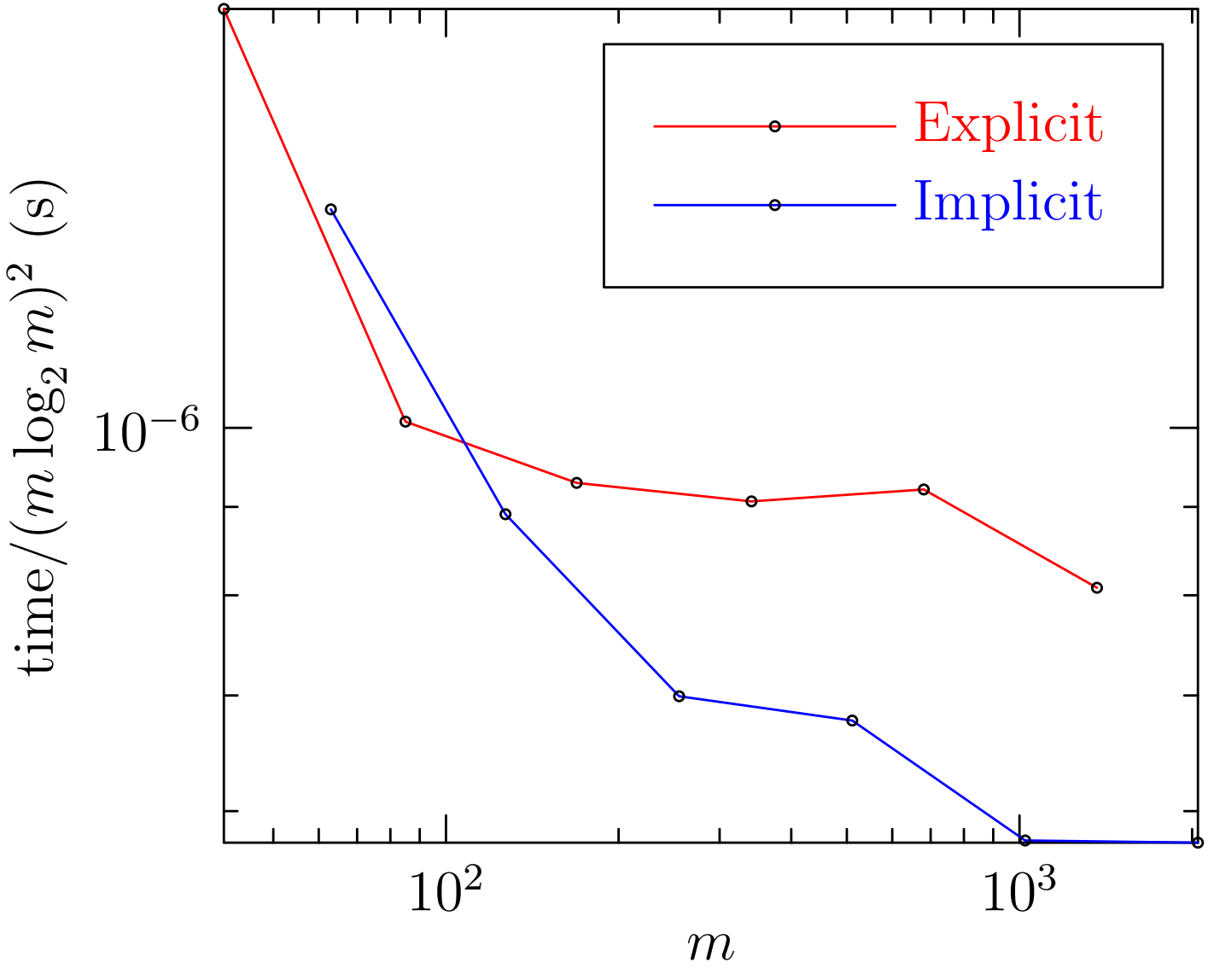
1D Implicit Hermitian Convolution



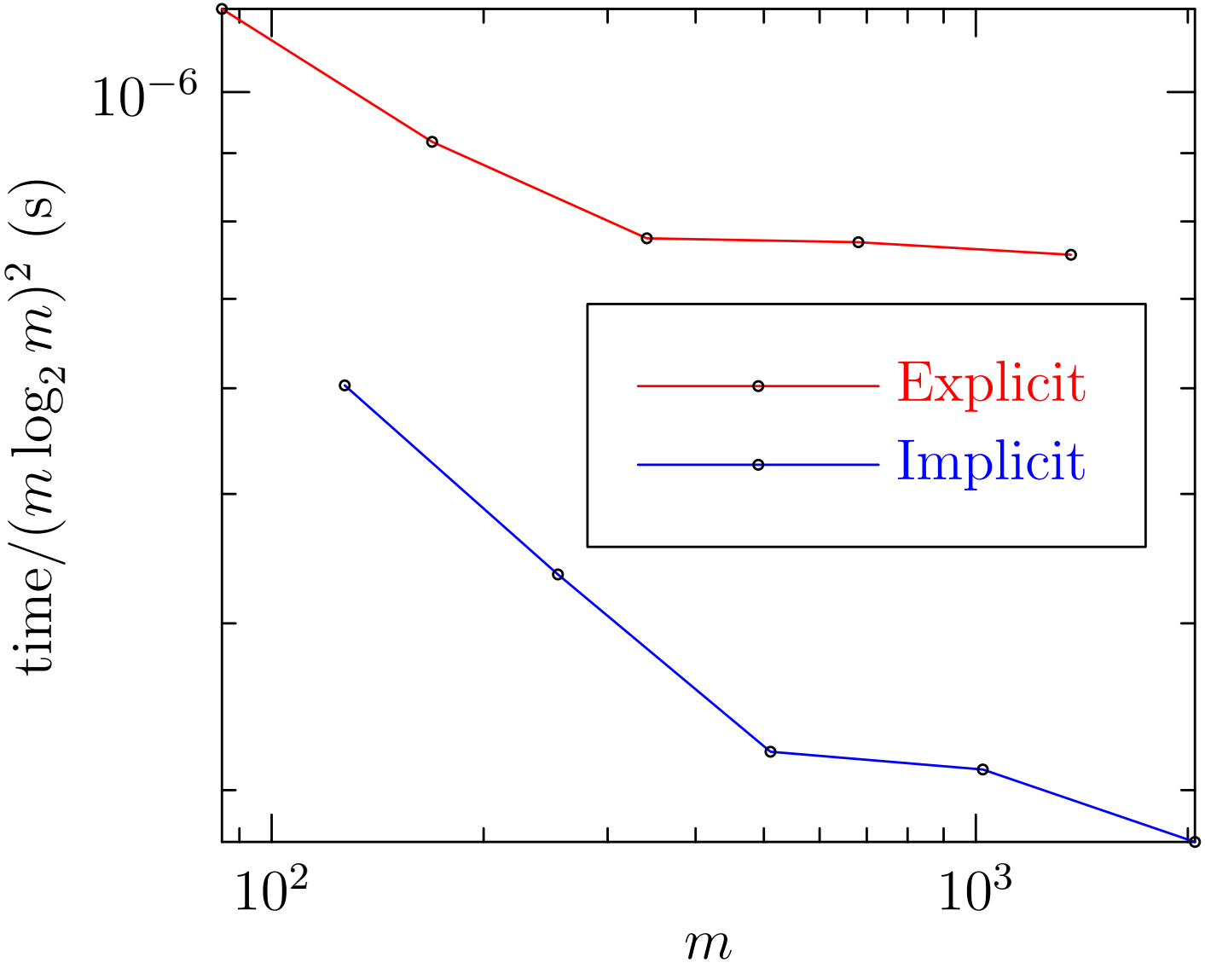
Shared-Memory Parallelization

- Our implicit and explicit convolution routines have been multithreaded for shared-memory architectures.

2D Navier–Stokes Pseudospectral [1 thread]



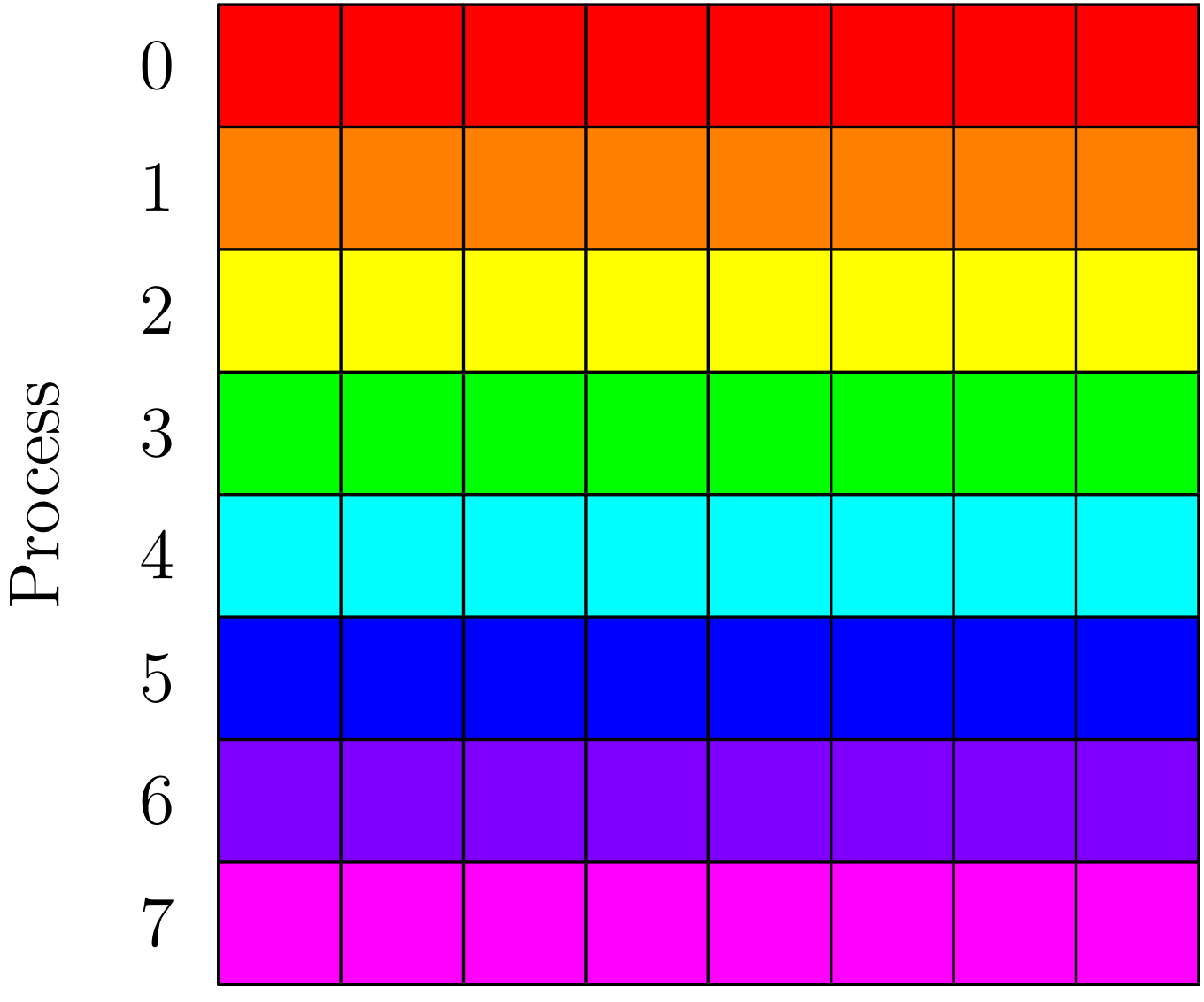
2D Navier–Stokes Pseudospectral [4 threads]



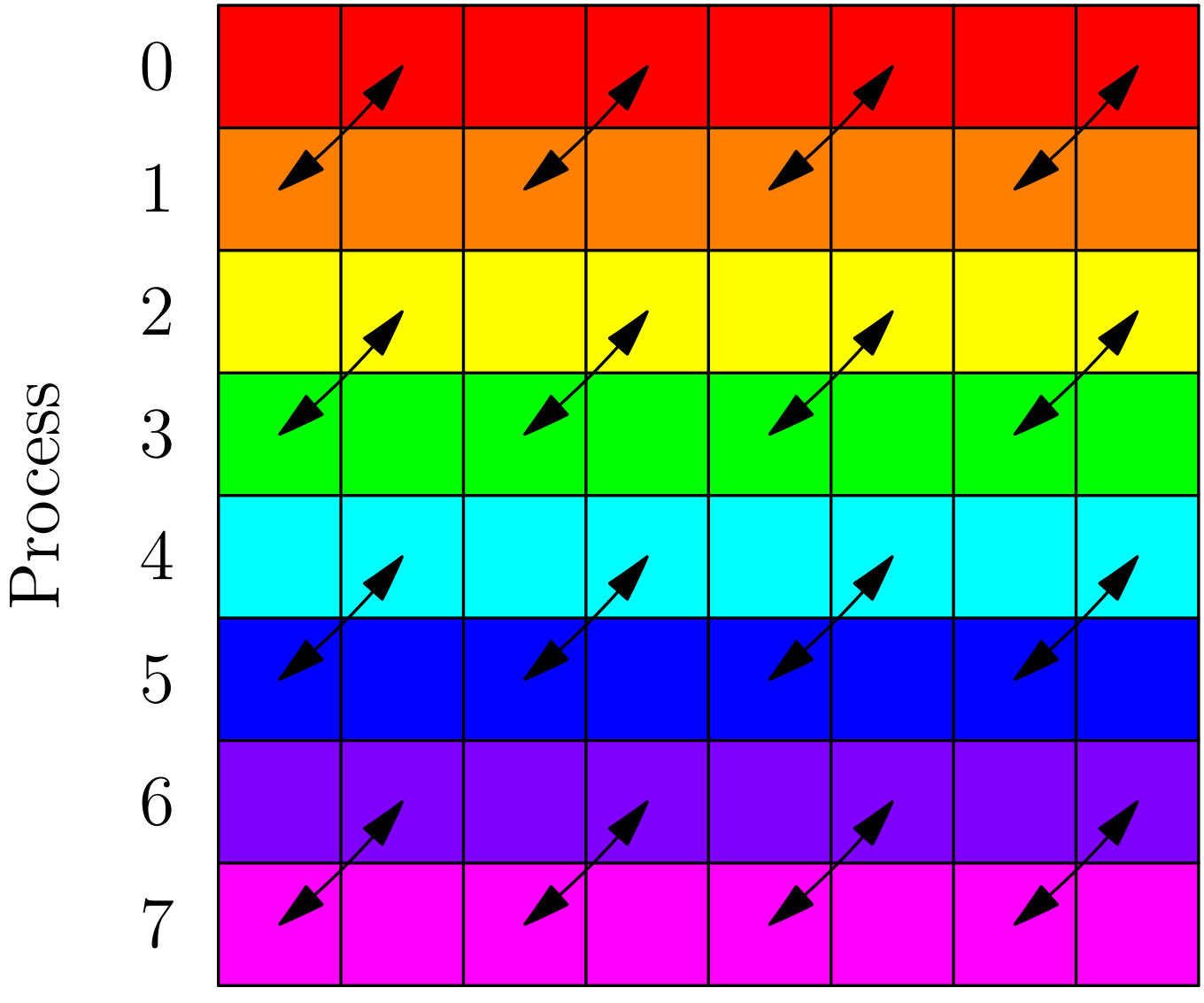
Distributed-Memory Parallelization

- The pseudospectral method uses a matrix transpose to localize the computation of the multi-dimensional FFTs onto individual processors.
- Parallel generalized slab/pencil decompositions have recently been developed for distributed-memory architectures.
- We have compared several distributed matrix transpose algorithms, both blocking and nonblocking, under pure MPI and hybrid MPI/OpenMP architectures.
- Local transposition is not required within a single MPI node.
- We have developed an adaptive algorithm, dynamically tuned to choose the optimal block size.

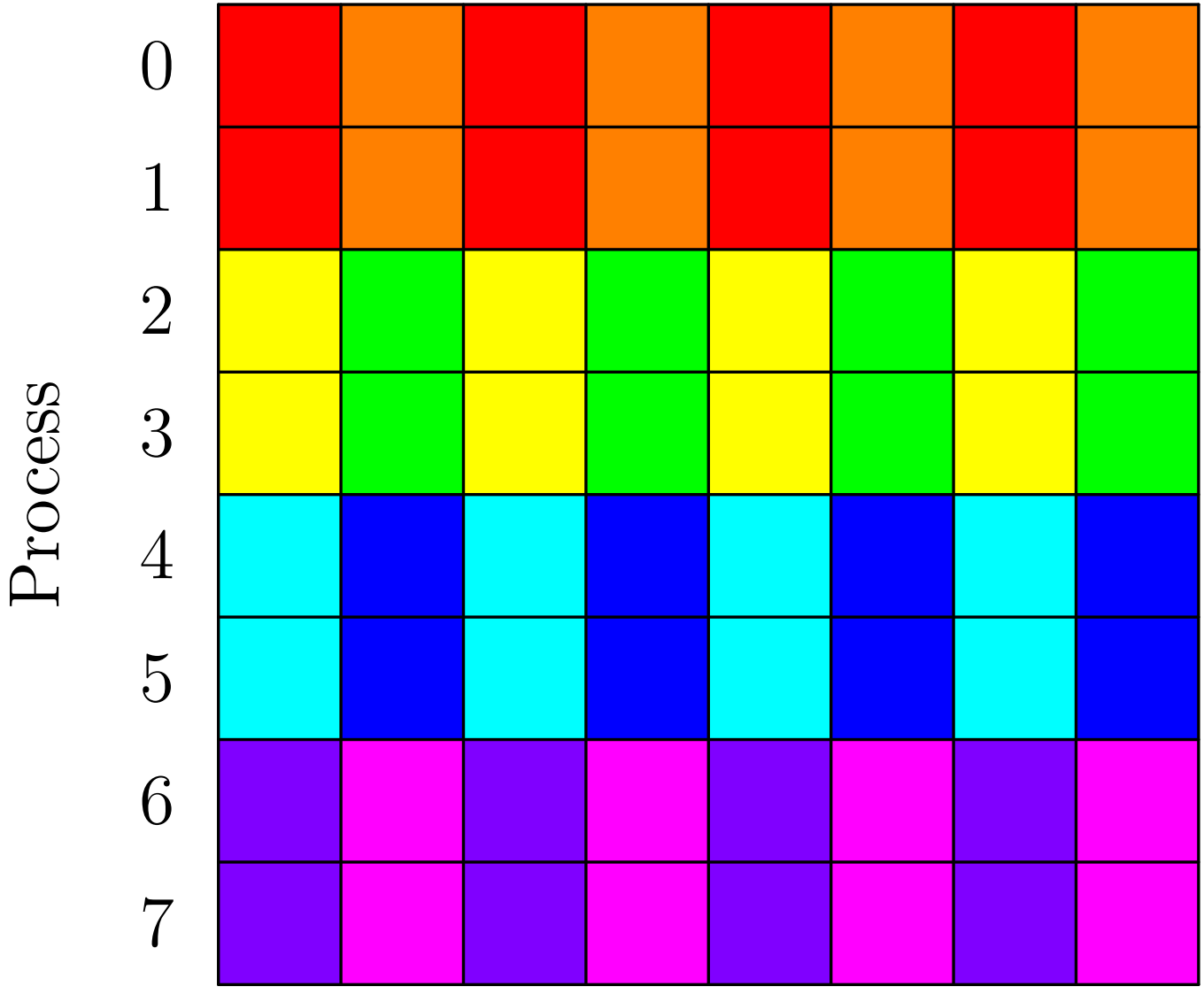
8×8 Block Transpose over 8 processors



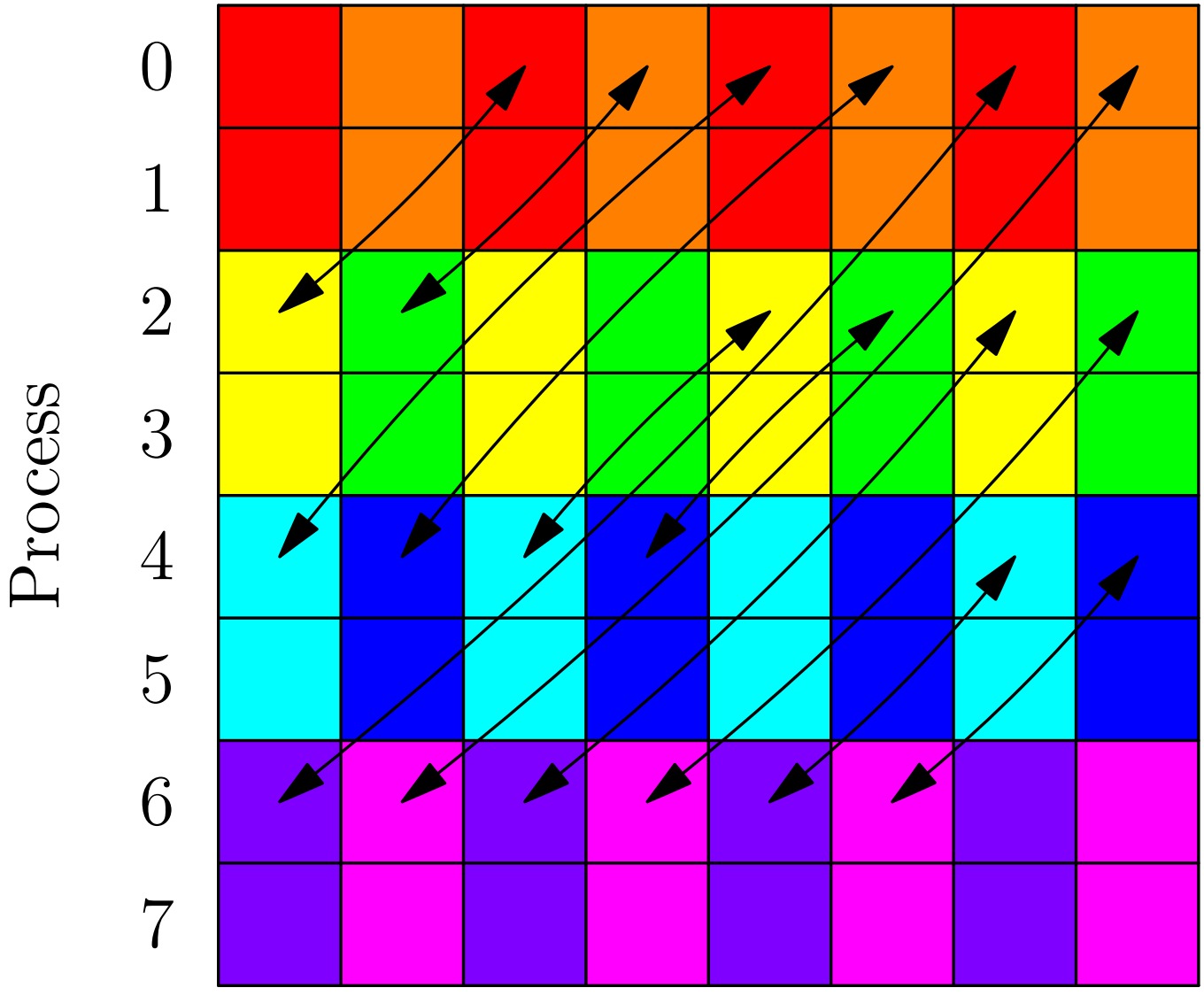
8×8 Block Transpose over 8 processors



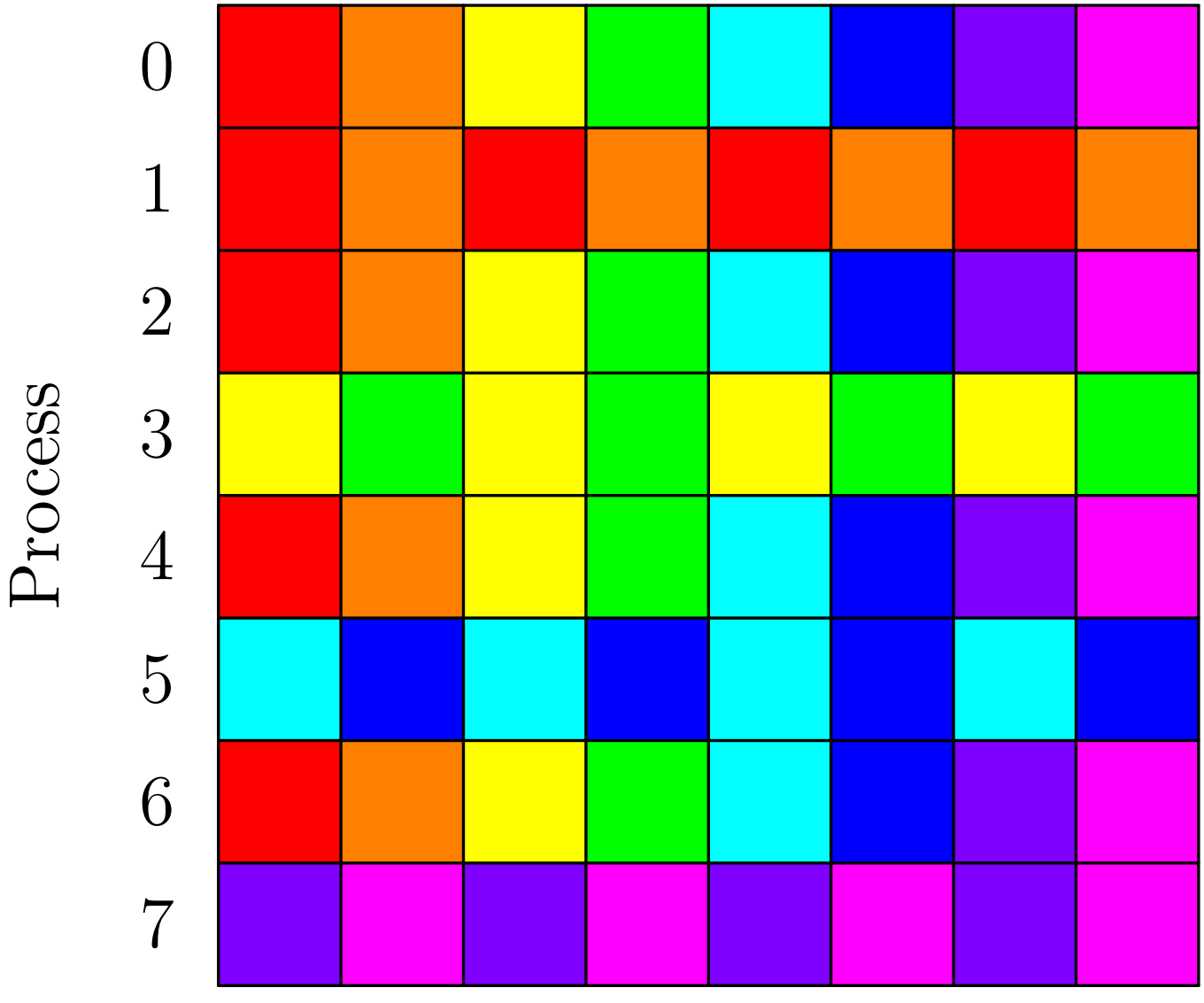
8×8 Block Transpose over 8 processors



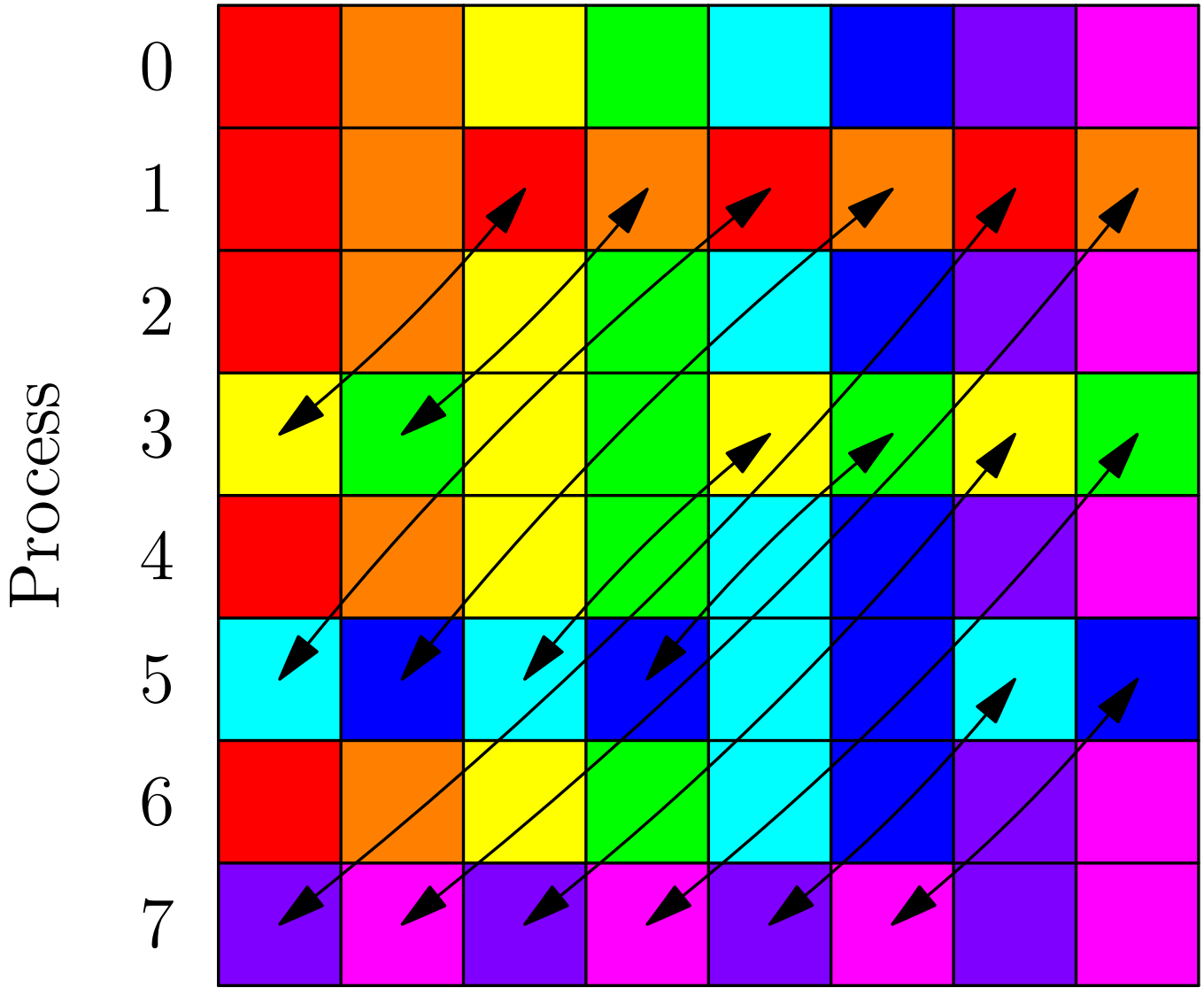
8×8 Block Transpose over 8 processors



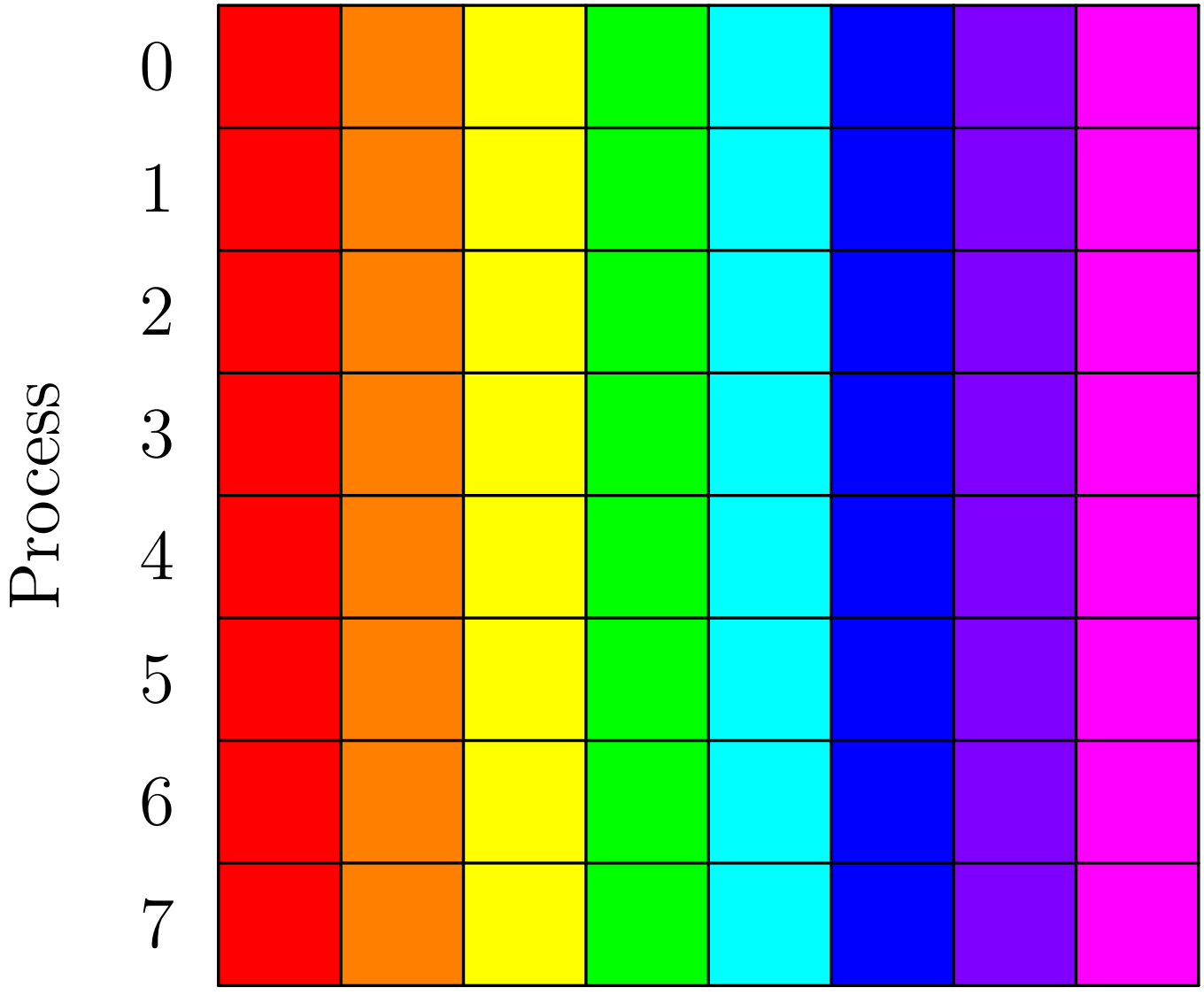
8×8 Block Transpose over 8 processors



8×8 Block Transpose over 8 processors



8×8 Block Transpose over 8 processors



Advantages of Hybrid MPI/OpenMP

- Use hybrid OpenMPI/MPI with the optimal number of threads:

  – yields larger communication block size;

  – local transposition is not required within a single MPI node;

  – allows smaller problems to be distributed over a large number of processors;

  – for 3D FFTs, allows for more slab-like than pencil-like models, reducing the size of or even eliminating the need for a second transpose.

  – sometimes more efficient (by a factor of 2) than pure MPI.

- The use of nonblocking MPI communications allows us to overlap computation with communication: this can yield up to an additional 32% performance gain for implicitly dealiased convolutions, for which a natural parallelism exists between communication and computation.

Communication Costs: Direct Transpose

- Suppose an $N \times N$ matrix is distributed over P processes with $P \mid N$.
- Direct transposition involves $P-1$ communications per process, each of size N^2/P^2, for a total per-process data transfer of

$$\frac{P-1}{P^2}N^2.$$

Block Transpose

- Let $P = ab$. Subdivide $N \times M$ matrix into $a \times a$ blocks each of size $N/a \times M/a$.
- Inner: Over each team of b processes, transpose the a individual $N/a \times M/a$ matrices, grouping all a communications with the same source and destination together.
- Outer: Over each team of a processes, transpose the $a \times a$ matrix of $N/a \times M/a$ blocks.

Communication Costs

- Let τ_ℓ be the typical latency of a message and τ_d be the time required to send each matrix element, so that the time to send a message consisting of n matrix elements is

$$\tau_\ell + n\tau_d$$

.

- The time required to perform a direct transpose is

$$T_D = \tau_\ell(P-1) + \tau_d \frac{P-1}{P^2} NM = (P-1)\left(\tau_\ell + \tau_d \frac{NM}{P^2}\right),$$

whereas a block transpose requires

$$T_B(a) = \tau_\ell\left(a + \frac{P}{a} - 2\right) + \tau_d\left(2P - a - \frac{P}{a}\right)\frac{NM}{P^2}.$$

- Let $L = \tau_\ell/\tau_d$ be the effective communication block length.

Direct vs. Block Transposes

- Since

$$T_D - T_B = \tau_d \left(P + 1 - a - \frac{P}{a} \right) \left(L - \frac{NM}{P^2} \right),$$

  we see that a direct transpose is preferred when $NM \geq P^2 L$, whereas a block transpose should be used when $NM < P^2 L$.

- To find the optimal value of a for a block transpose consider

$$T'_B(a) = \tau_d \left(1 - \frac{P}{a^2} \right) \left(L - \frac{NM}{P^2} \right).$$

- For $NM < P^2 L$, we see that T_B is convex, with a minimum at $a = \sqrt{P}$.

Optimal Number of Threads

- The minimum value of T_B is

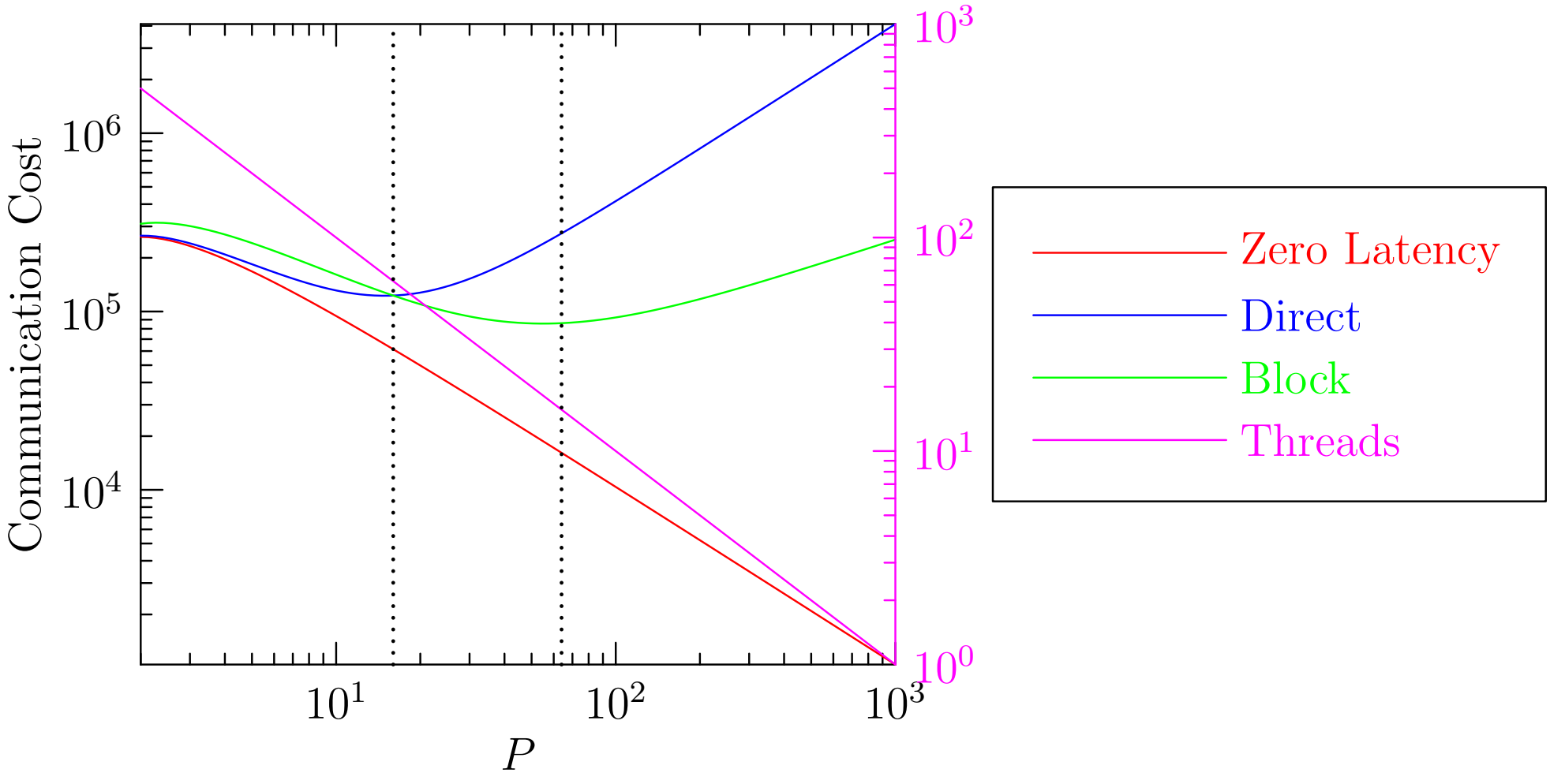
$$T_B(\sqrt{P}) = 2\tau_d\left(\sqrt{P}-1\right)\left(L+\frac{NM}{P^{3/2}}\right)$$
$$\sim 2\,\tau_d\sqrt{P}\left(L+\frac{NM}{P^{3/2}}\right), \qquad P \gg 1.$$

- The global minimum of T_B over both a and P occurs at

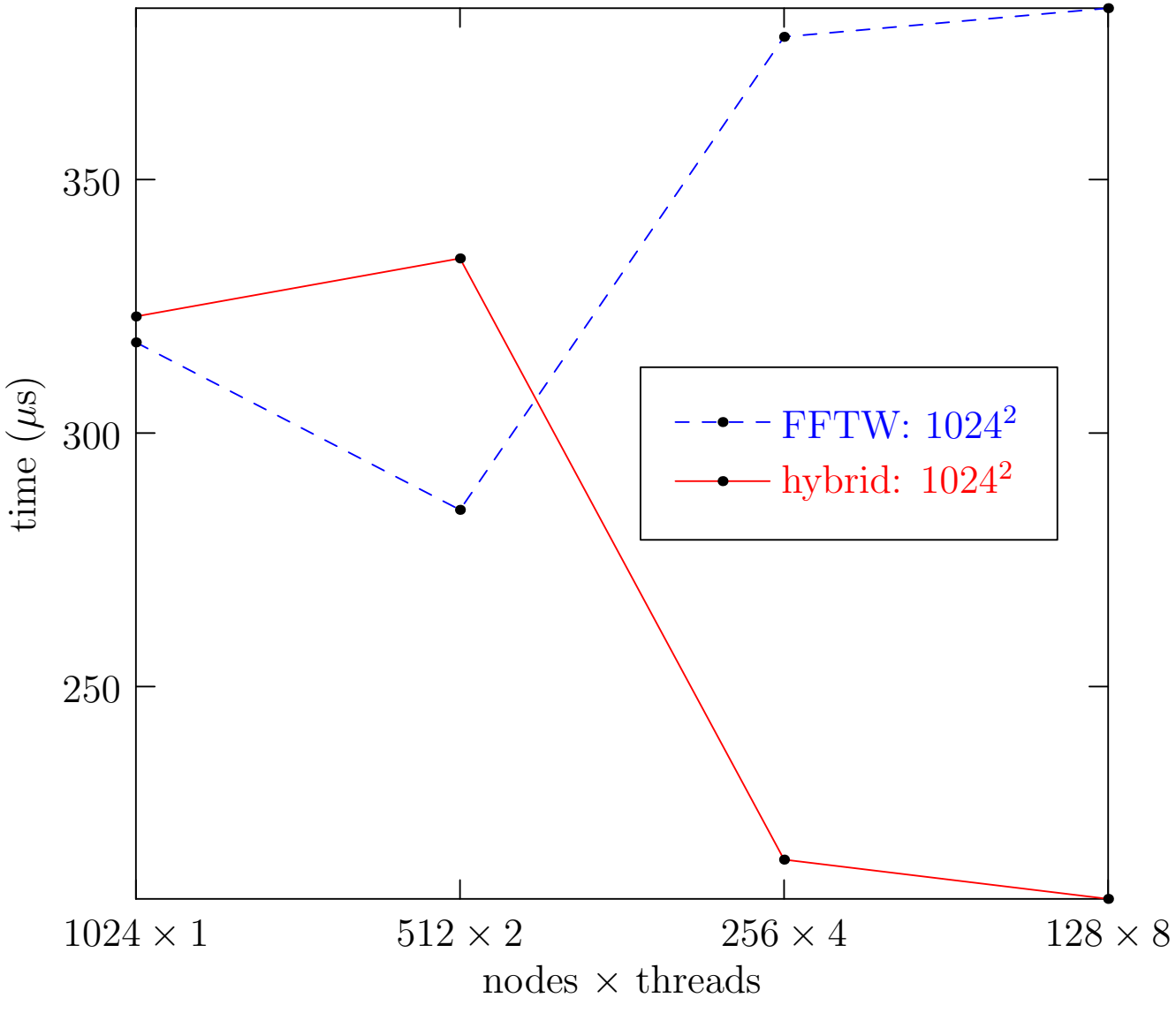
$$P \approx (2NM/L)^{2/3}.$$

- If the matrix dimensions satisfy $NM > L$, as is typically the case, this minimum occurs above the transition value $(NM/L)^{1/2}$.

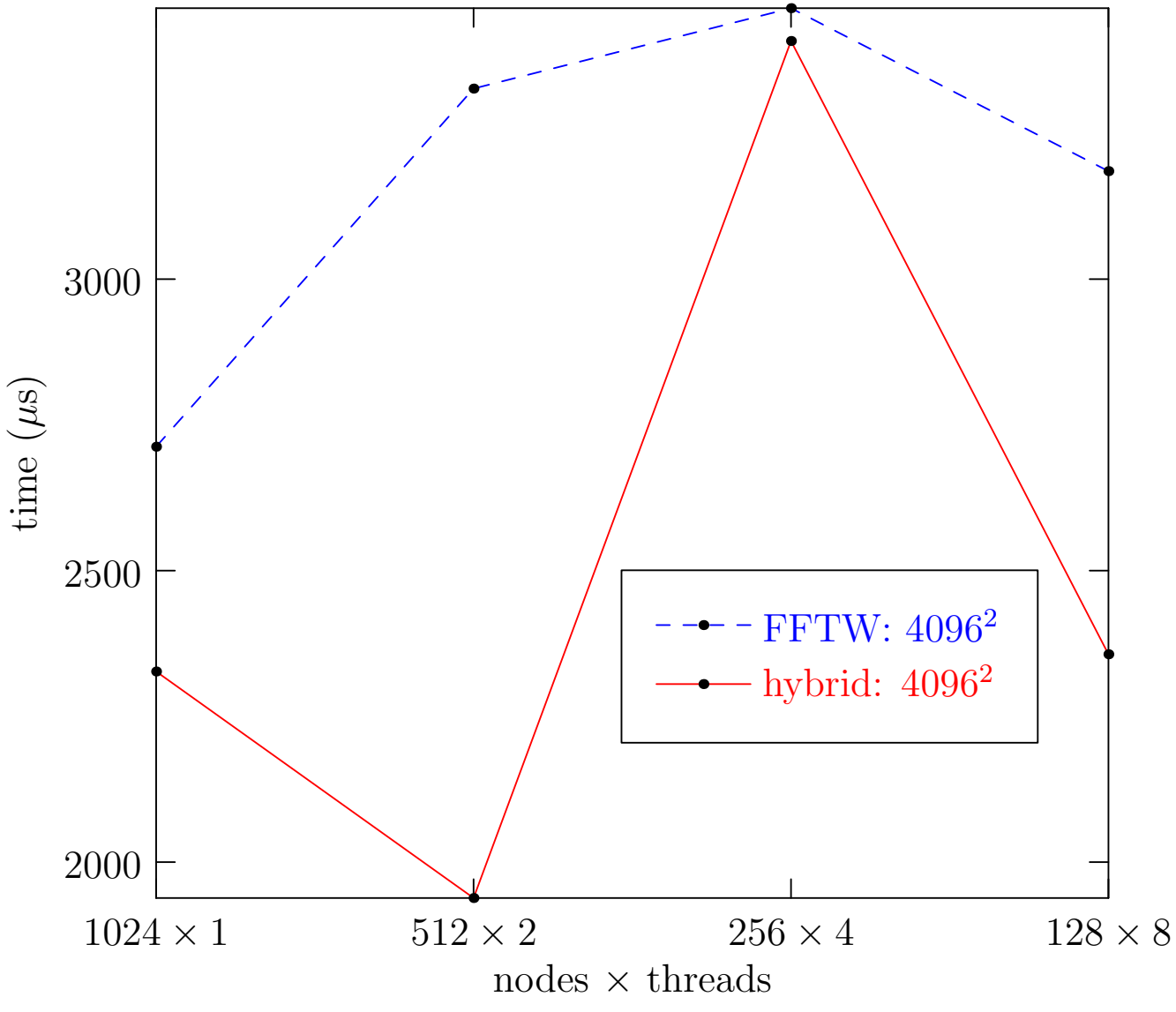
Transpose Communication Costs



Optimal Number of Threads: 1024×1024



Optimal Number of Threads: 4096×4096



Windowed Time Averages

- Suppose we evolve an equation of the form

$$\frac{d\omega_{\boldsymbol{k}}}{dt} = S_{\boldsymbol{k}}.$$

- Once a statistically stationary state has been achieved, we may want to compute windowed time averages of moments like $|\omega_{\boldsymbol{k}}|^n$ and $S_{\boldsymbol{k}}\omega^*_{\boldsymbol{k}}$.

- But the saturation time is not normally known until after the simulation is completed.

Dynamic Moment Averaging

- Advantageous to precompute time-integrated moments like

$$M_n(t) = \int_0^t |\omega_{\boldsymbol{k}}(\tau)|^n \, d\tau.$$

- This can be accomplished done by evolving

$$\frac{dM_n}{dt} = |\omega_{\boldsymbol{k}}|^n,$$

  along with the vorticity $\omega_{\boldsymbol{k}}$ itself, using *the same* temporal discretization.

- These evolved quantities M_n can be used to extract accurate statistical averages during the post-processing phase, once the saturation time t_1 has been determined by the user:

$$\int_{t_1}^{t_2} |\omega_{\boldsymbol{k}}|^n (\tau) \, d\tau = M_n(t_2) - M_n(t_1).$$

Enstrophy Balance

$$\frac{\partial \omega_{\boldsymbol{k}}}{\partial t} + \nu k^2 \omega_{\boldsymbol{k}} = S_{\boldsymbol{k}} + f_{\boldsymbol{k}},$$

- Multiply by $\omega^*_{\boldsymbol{k}}$ and integrate over wavenumber angle $\Rightarrow$ enstrophy spectrum $Z(k)$ evolves as:

$$\frac{\partial}{\partial t} Z(k) + 2\nu k^2 Z(k) = 2T(k) + G(k),$$

where $T(k)$ and $G(k)$ are the corresponding angular averages of $\mathrm{Re}\,\langle S_{\boldsymbol{k}} \omega^*_{\boldsymbol{k}} \rangle$ and $\mathrm{Re}\,\langle f_{\boldsymbol{k}} \omega^*_{\boldsymbol{k}} \rangle$.

Nonlinear Enstrophy Transfer Function

$$\frac{\partial}{\partial t} Z(k) + 2\nu k^2 Z(k) = 2T(k) + G(k).$$

- Let

$$\Pi(k) \doteq 2 \int_k^\infty T(p)\, dp$$

represent the nonlinear transfer of enstrophy into $[k, \infty)$.

- Integrate from k to ∞:

$$\frac{d}{dt} \int_k^\infty Z(p)\, dp = \Pi(k) - \epsilon_Z(k),$$

where $\epsilon_Z(k) \doteq 2\nu \int_k^\infty p^2 Z(p)\, dp - \int_k^\infty G(p)\, dp$ is the total enstrophy transfer, via dissipation and forcing, out of wavenumbers higher than k.

- A positive (negative) value for $\Pi(k)$ represents a flow of enstrophy to wavenumbers higher (lower) than k.
- When $\nu = 0$ and $f_{\boldsymbol{k}} = 0$:

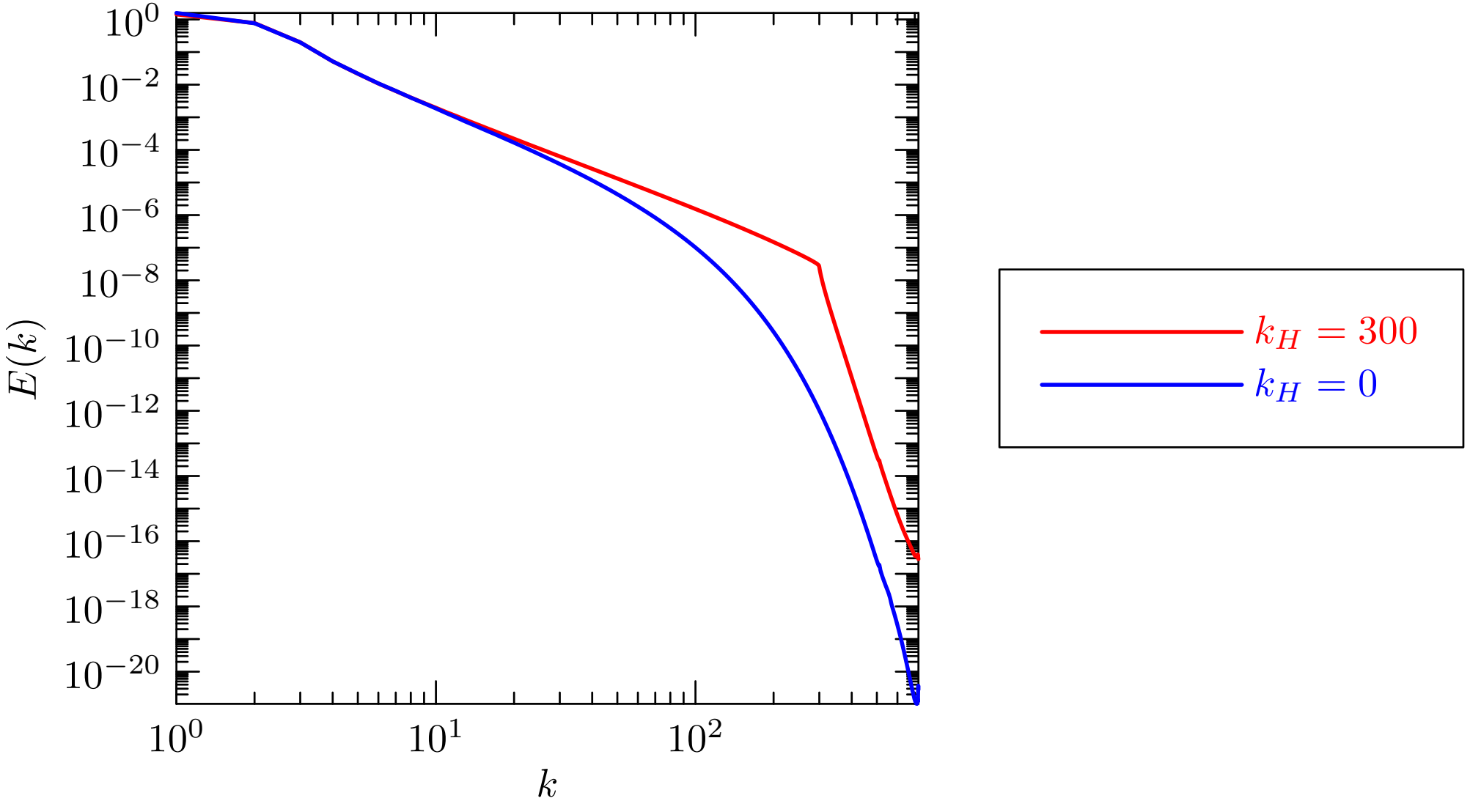
$$0 = \frac{d}{dt} \int_0^\infty Z(p)\, dp = 2 \int_0^\infty T(p)\, dp,$$

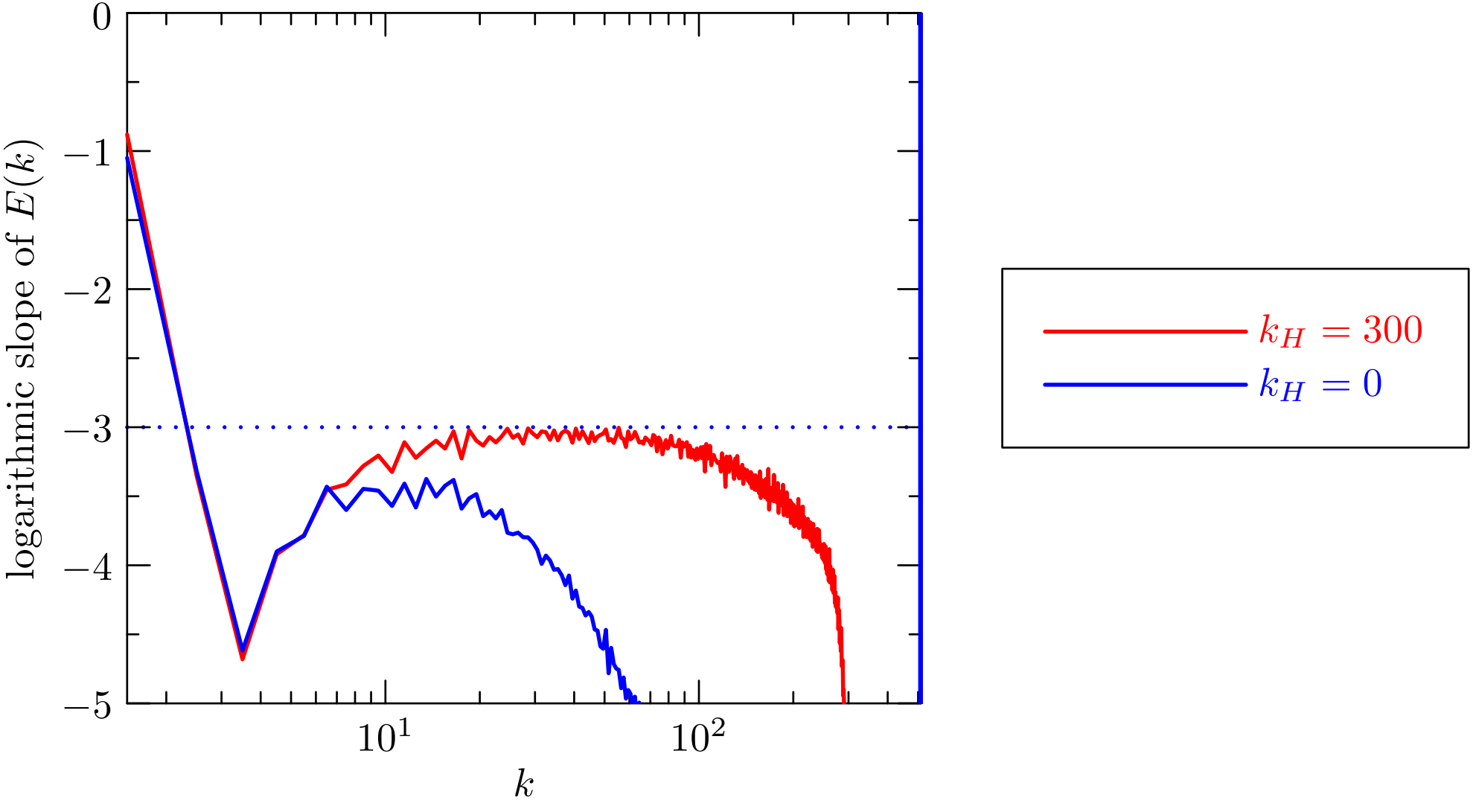
so that

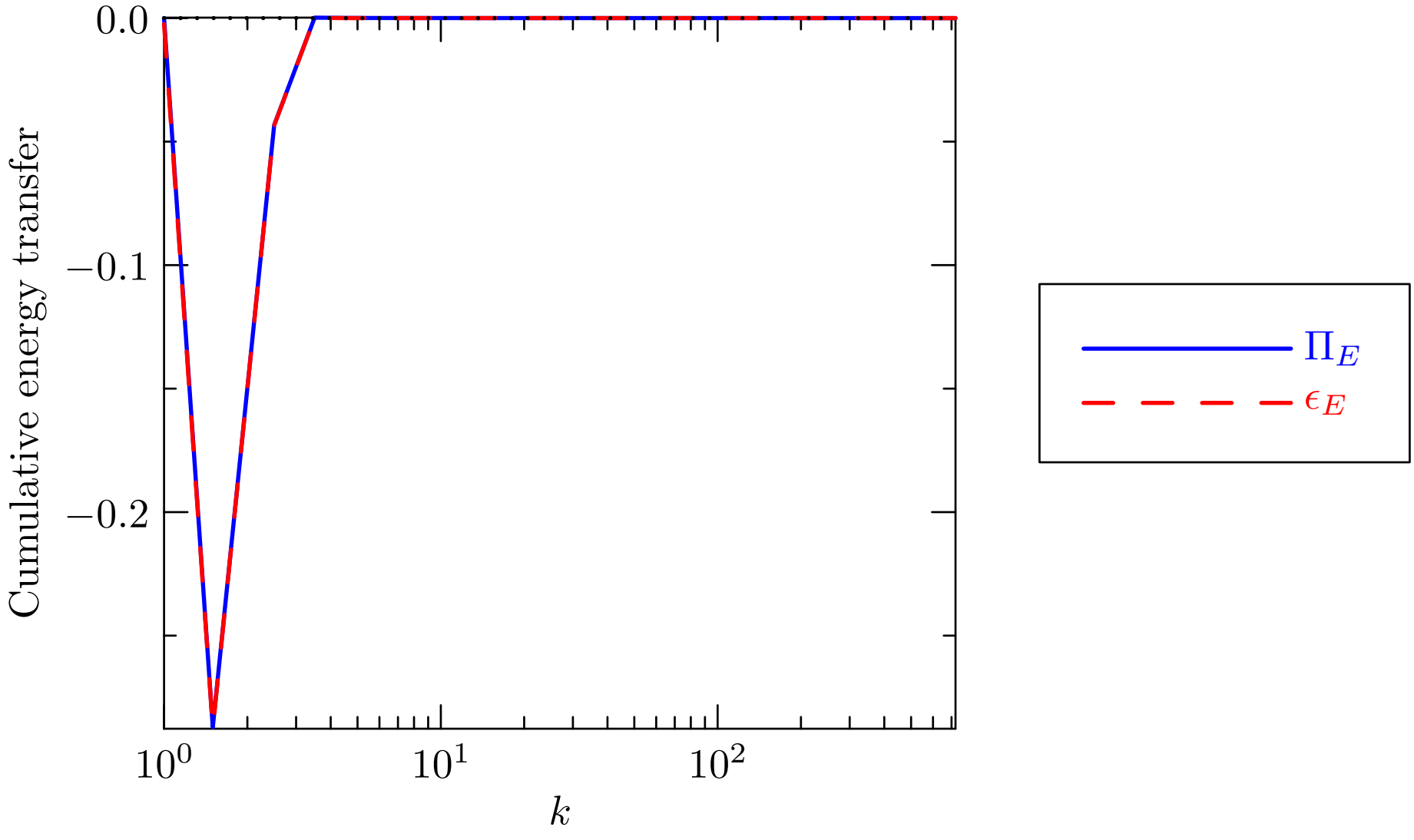
$$\Pi(k) = 2 \int_k^\infty T(p)\, dp = -2 \int_0^k T(p)\, dp.$$

- Note that $\Pi(0) = \Pi(\infty) = 0$.
- In a steady state, $\Pi(k) = \epsilon_Z(k)$.
- This provides an excellent numerical diagnostic for determining the saturation time t_1.

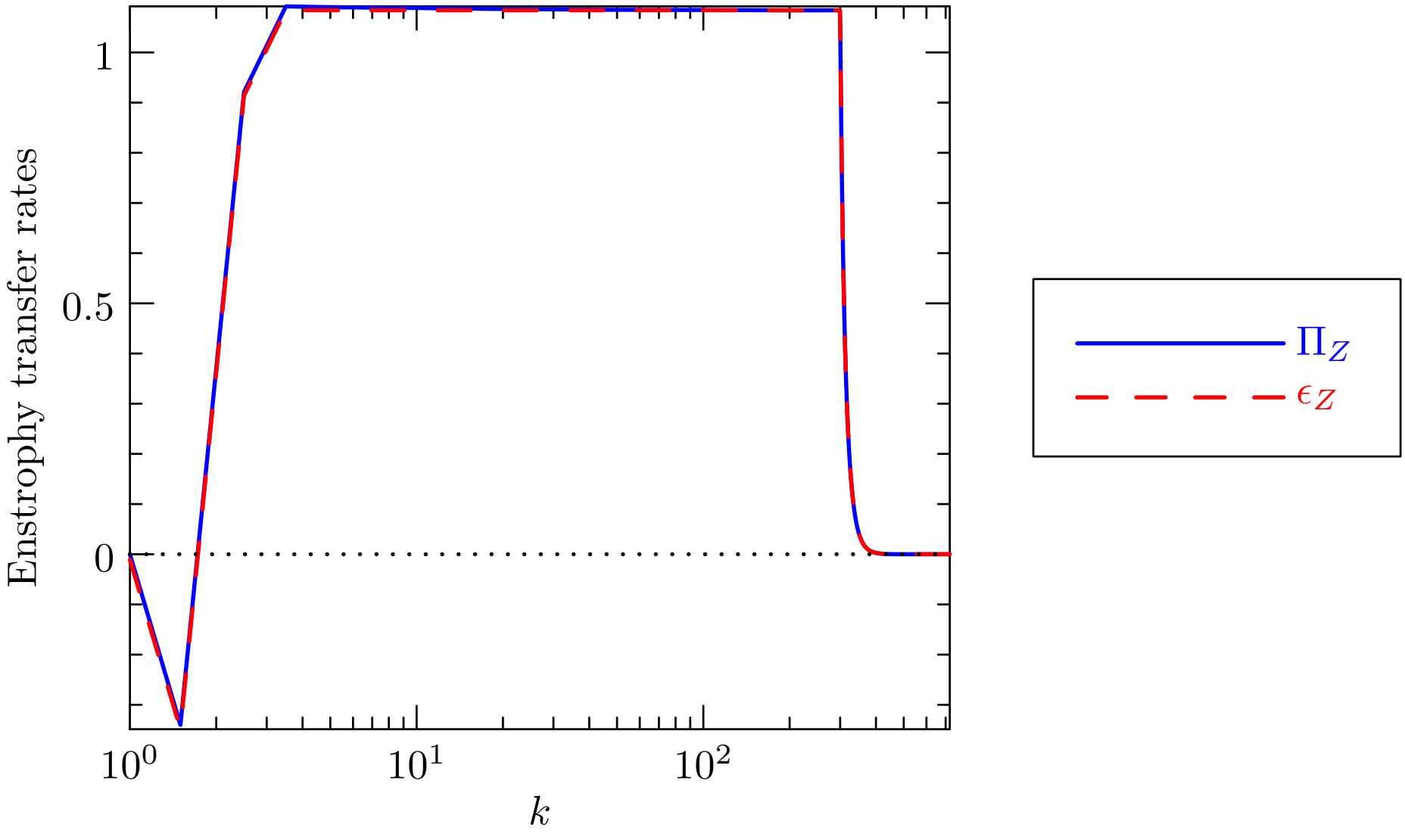
Forcing at $k = 2$, friction for $k < 3$, viscosity for $k \geq k_H = 300$ (1023×1023 dealiased modes)



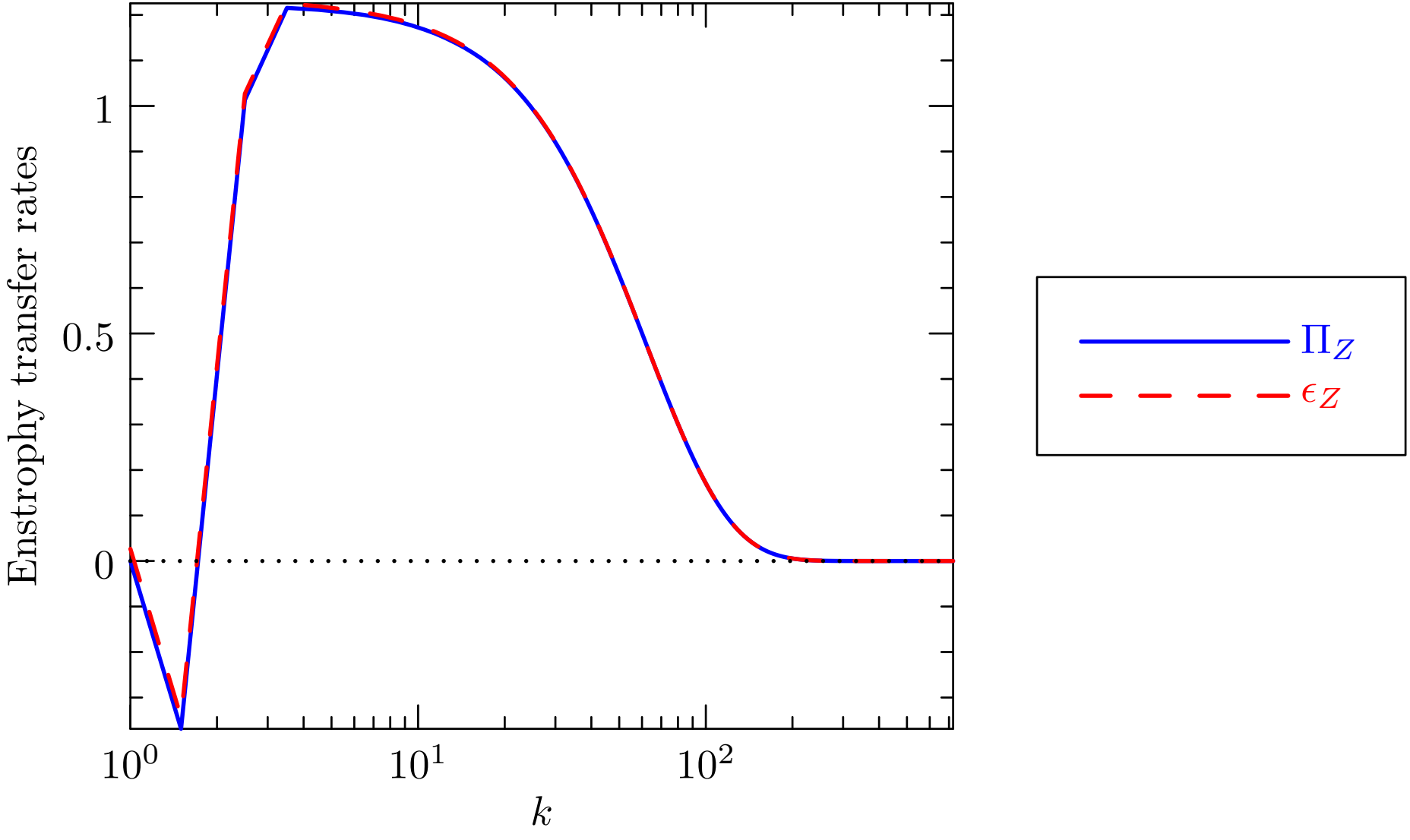

0
−1
−2
−3
−4
−5
logarithmic slope of $E(k)$
10^1
10^2
k
$k_H = 300$
$k_H = 0$



Cutoff viscosity ($k \geq k_H = 300$)



Cutoff viscosity ($k \geq k_H = 300$)



Molecular viscosity ($k \geq k_H = 0$)

Conservative Integration

- Conservative integration [SIAM J. Appl. Math 59, 1112 (1999)] provides a useful diagnostic technique for ensuring that the underlying dynamical symmetries have been correctly implemented.

Conclusions

- For centered convolutions in d dimensions implicit padding asymptotically uses $(2/3)^{d-1}$ of the conventional storage.
- The factor of 2 speedup is largely due to increased data locality.
- Highly optimized and parallelized implicit dealiasing routines have been implemented as a software layer `FFTW++` (v 2.00) on top of the `FFTW` library and released under the Lesser GNU Public License: `http://fftwpp.sourceforge.net/`
- The advent of implicit dealiasing of convolutions makes overlapping transposition with FFT computation feasible.
- Dynamic moment averaging allows the integration time window to be specified by the user *a posteriori*. The cumulative enstrophy transfer function is an excellent diagnostic for determining the saturation time.
- Writing of a high-performance dealiased pseudospectral code is now a relatively straightforward exercise!

References

[Bowman & Roberts 2011] J. C. Bowman & M. Roberts, SIAM J. Sci. Comput., **33**:386, 2011.

[Bowman & Roberts 2013] J. C. Bowman & M. Roberts, submitted to Parallel computing, 2013.

[Orszag 1971] S. A. Orszag, Journal of the Atmospheric Sciences, **28**:1074, 1971.

[Shadwick *et al.* 1999] B. A. Shadwick, J. C. Bowman, & P. J. Morrison, SIAM J. Appl. Math., **59**:1112, 1999.