10 years of ROMS project:
An overview with emphasis of what is overlooked, overdue, or missing.

Alexander Shchepetkin, I.G.P.P. UCLA

Los Angeles, CA, October 2007

August 24, 1997 as starting date for ROMS project: **We are 10 years old now**

Acronym "ROMS" (a.k.a. Regional Oceanic Modeling System) was introduced by Dale Haidvogel in October 1998 during Davis Meeting

ROMS evolved (as opposite to *intelligent design*) with full range of consequences of this process:

- from Hernan's SCRUM 3.0 code
- mess, claims, controversy, scrutiny...
- *absence of claims*, ...sometimes
- inheritance, legacies, sometimes irrational choices, branches
- **there is more than one code called ROMS**

Linux-like-spirited community with multiple development centr*e*s

- exchange of ideas, mutual help network
- voices to unify and make one "official" code
- go elsewhere and **google** for janjic +skamarock +"white paper"
  `www.emc.ncep.noaa.gov/mmb/papers/janjic/response2/response.html`
  `www.mmm.ucar.edu/people/skamarock/black_janjic_spectra_response.pdf`
  (viewer discretion advised)

ROMS have survived

What is ROMS?

- an ocean modeling code
- **by vertical coordinate:** $\in$ σ-class models, but code stores $z(x, y, \sigma)$ as an array $\Rightarrow$ a rather general vertical coordinate code.
- **by time stepping engine:** free-surface, split-explicit, barotropic-baroclinic mode coupling algorithm
- always intended for limited-area modeling $\Rightarrow$ open boundary conditions are in focus
- Boussinesq approximation, although not...
- hydrostatic, although ...
- orthogonal curvilinear grid in horizontal
- built around new time-stepping algorithms for hyperbolic system equations; always use forward-backward feedback
- higher than second-order accuracy spatial discretization for critical terms: advection, pressure-gradient...
- grid nesting, `RomsTools` (more than one branch)
- adjoint
- ground-up design philosophy, focusing on multi-component interplay of remotely-related features and algorithms: code infrastructure is distinct from *modular* (like in MOM/POP) design
- parallel via 2D domain decomposition: threaded/OpenMP, or MPI, or both; multiple computer architecture support
- code architecture decisions involve optimization in multidimensional space, including model physics, numerical algorithms, computational performance and cost
- coupled with sub-models (biology, sediment transport, etc...)

Dealing with consequences: outstanding, overlooked, overdue issues

- Generalized Forward-Backward barotropic mode at least 2.5:1 outperforms the old LF-AM3 version. It finally made it in AGRIF ROMS ... after 4.5 years. Still to make it into Rutgers ROMS. Barotropic mode consumes 25%...40% of CPU time for typical ROMS configuration. The longest delay ever.
- Laurent Debreu reported a source of mode-splitting error due to crude startup algorithm of barotropic mode. The subsequent changes lead to a more accurate and robust splitting. Tide simulation without reverting to a more dissipative filter for fast-time-averaging. Not yet in Rutgers code.
- Pressure gradient scheme is only implemented only 50% in Rutgers ROMS: the *monotonicity* of interpolated stratification is *not guaranteed* because adiabatic density differencing is not used. There is also related update about compressibility effect in seawater EOS presented in this talk.
- Compiler issues with advanced features of F90: CPU performance of older version of Rutgers ROMS 1.8/1.9 is as much as 1.4 times better than of the later 2.2 and 3.0 versions in OpenMP mode (1.8/1.9 were not MPI-capable codes), if using Intel compiler, especially on an Intel CPU (other compilers produce less contrast, but they result in slower executables as well. This is a known/noticed problem and it does not go away by itself "as compilers improve". The whole community accepted significant performance hit, but only handful of people complained. There is also pressure from ESMF to mandate code conventions some of which disregard performance
- OpenMP Standard specification has been changed since 2001. Compilers follow $\Rightarrow$ code which worked correctly for several years previously may not work correctly.
- MPI code allows OpenMP threads within MPI subdomains at last

Selected topics:

EOS: Compressibility effects and Boussinesq approximation

Flather Boundary Conditions: an update

Open MP Standard

Linux Cluster computing: caveats and experiences

EOS Compressibility effects in Boussinesq model

Boussinesq approximation: rewrite equations to make density be multiplier,

$$\partial_t (\rho \mathbf{u}) + ... = \nabla P + ... \qquad P = g \int_z^{\zeta} \rho \, dz' + ... \text{ [NH-terms, if any]}$$

and replace $\rho \rightarrow \rho_0$ everywhere, except where multiplied by g.

- relies on smallness of variations of density $\sim 5\%$
- Boussinesq approximation is often associated with replacing *in situ* pressure inside EOS with bulk pressure $P = g\rho_0|z|$, which is basically depth
- assumes that water is *dynamically incompressible* $\Rightarrow$ no acoustic waves; density appears only in context of *buoyancy anomaly*, $-g\rho'/\rho_0$; simplifies a lot especially in non-hydrostatic case;
- nevertheless realistic *compressible* UNESCO-type EOS is used in ocean modeling because it is physically important to capture thermobaric effect
- **theoreticians love to write papers about:** McDougall & Garret, 1992; Jackett & McDougall, 1997; Dewar, Hsueh, McDougall, & Yuan, 1998; Lu,2000; Greatbatch, 2001; de Szoeke & Samelson, 2002; McDougall, Greatbatch, & Lu., 2002; Greatbatch & McDougall, 2003; Losch, Adcroft, & Campin, 2004; and counting...
- is believed to admit $\sim 5\%$ errors in velocities, transport
- it is a big deal, look at internal inconsistencies...
- no, it is not a big deal, because of *isomorphism* between Boussinesq equations and non-Boussinesq in pressure coordinates: one can simply *reinterpret* Boussinesq model results as non-Boussinesq (hydrostatic version for both)

An example of internal inconsistency of Boussinesq model with compressible EOS: Assuming spatially uniform $\Theta, S = $ const, perturbed free surface, and hydrostatic balance,

$$\rho = \rho(P) = \rho\Big|_{P=0} + P/c^2, \qquad \partial_z P = -g\rho, \quad \text{along with} \quad P\Big|_{z=\zeta} = 0$$

which remaps $P \leftrightarrow z$ as

$$\rho = \rho\Big|_{z=\zeta} \exp\left\{g\frac{\zeta - z}{c^2}\right\} \approx \rho\Big|_{z=\zeta} + g\rho_0\frac{\zeta - z}{c^2}$$

[c is speed of sound, $1/c^2 = \partial\rho/\partial P$, and for simplicity we assume smallness $g|z|/c^2 \ll 1$ which is not principal] The acceleration created PGF due to perturbation in free surface is then

$$-\frac{1}{\rho_0}\nabla_x P = -\frac{g\,\rho|_{z=\zeta}}{\rho_0} \cdot \nabla_x \int\limits_z^\zeta \exp\left\{g\frac{\zeta - z'}{c^2}\right\} dz' \approx -g\left[\frac{\rho|_{z=\zeta}}{\rho_0} + \frac{g(\zeta - z)}{c^2}\right]\nabla_x\zeta$$

increases with depth.

non-Boussinesq answer: acceleration is $-g\nabla_x\zeta$ **independent** from depth; this is **an exact** result, even if $g|z|/c^2$ is not small, and even in the case when $\rho = \rho(P)$ is nonlinear. Simply put, in comparison with constant-density case **both** ρ and $\nabla_x P$ are multiplied by

$$r(z) = \exp\left\{g\frac{\zeta - z}{c^2}\right\}$$

which cancels out when computing acceleration $-(1/\rho)\nabla_x P$

Boussinesq approximation retains $r(z)$ in one place but neglects it in the other, resulting in spurious vertical shear in PGF acceleration.

How does this affect ROMS?

- in computation of pressure gradient force: if EOS of JM95 (UNESCO-type) is used, the resultant r.h.s. for 3D momenta fully contains spurious vertical shear

- when computing

$$\overline{\rho} = \frac{1}{D}\int_{-h}^{\zeta} \rho\, dz \quad \text{and} \quad \rho^* = \frac{2}{D^2}\int_{-h}^{\zeta} \left\{ \int_{z}^{\zeta} \rho\, dz' \right\} dz \,, \qquad D = h + \zeta$$

  for the use in barotropic-mode pressure gradient, the resultant $\overline{\rho}$ and ρ^* are related as it would be for a stratified water column, when in fact, physically there is no stratification [e.g., $\rho = \rho(P)$ case];

- EOS is computed in *slow time*, but contains traces of free surface signal, which is kept unchanged during fast-time stepping $\Rightarrow$ contribution to mode splitting error. The contribution is very small, but because free-surface field available to EOS is taken from previous time step, it results in effectively Forward Euler stepping for these terms. This kind of instability was first observed in POM and reported by Robinson, Padman, & Levine, 2001: A correction to the baroclinic pressure gradient term in the Princeton ocean model. *J. Atmos. Ocean. Technol.*, **18**, pp. 1068-1075 [although they did not classify it as mode-splitting instability]. Their proposed remedy is to suppress compressibility effects in EOS altogether.

- Griffies, 2002, advocates to abandon Boussinesq approximation, incl. the use of *in situ* pressure inside EOS. Although this eliminates spurious shear, it does not fix mode splitting (one must somehow exclude influence of free surface in EOS, which contradicts the idea of *in situ* pressure; or redesign barotropic mode; implicit stepping for free-surface is immune to this because it is too dissipative). Something remains to be done about *adiabatic differencing* (discussed below)

Three are 4 reasons why a Boussinesq ocean model needs EOS:

- computation of pressure gradient force;
- evaluation of stability of stratification as well as stability of external thermodynamic forcing (buoyancy flux) needed for mixing and planetary boundary layer parameterization;
- computation of slopes of neutral surfaces need by horizontal (along isopycnals) diffusion
- computing of $\overline{\rho}$ (vertically averaged density) and ρ^* (normalized vertically averaged pressure),which participate in barotropic–baroclinic mode splitting.

the role of EOS is to translate gradients of T, S into gradients of density

in situ density is not needed

PGF in sigma-coordinates needs

$$\mathcal{J}_{x,s}(\rho,z) = -\alpha\cdot\mathcal{J}_{x,s}(\Theta,z) + \beta\cdot\mathcal{J}_{x,s}(S,z)$$

where $\alpha = \alpha(\Theta,S,P) = -\left.\dfrac{\partial\rho}{\partial\Theta}\right|_{S,P=\text{const}}$ $\beta = \beta(\Theta,S,P) = \left.\dfrac{\partial\rho}{\partial S}\right|_{\Theta,P=\text{const}}$

alternatively, if

$$\rho = \rho_1^{(0)} + \rho_1'(\Theta,S) + \sum_{m=1}^{n}\left(q_m^{(0)} + q_m'(\Theta,S)\right)\cdot|z|^m$$

then $\rho_1^{(0)}$, $q_m^{(0)}$-terms are all out:

$$\mathcal{J}_{x,s}(\rho,z) = \mathcal{J}_{x,s}(\rho_1',z) + \sum_{m=1}^{n}\mathcal{J}_{x,s}(q_m',z)\cdot|z|^m$$

adiabatic differencing

$$\Delta\rho'^{(\text{ad})}_{i+\frac{1}{2},j,k} = \rho'_{1\,i+1,j,k} - \rho'_{1\,i,j,k} + \sum_{m=1}^{n}\left(q'_{m\,i+1,j,k} - q'_{m\,i,j,k}\right)\left|\frac{z_{i+1,j,k}+z_{i,j,k}}{2}\right|^m$$

the two adjacent adiabatic differences are averaged using harmonic mean, and (if needed) the compressible part is computed and added separately,

$$d_{i,j,k} \equiv \left.\frac{\partial\rho}{\partial\xi}\right|_{i,j,k} = \frac{1}{\Delta\xi}\cdot\frac{2\Delta\rho'^{(\text{ad})}_{i+\frac{1}{2},j,k}\cdot\Delta\rho'^{(\text{ad})}_{i-\frac{1}{2},j,k}}{\Delta\rho'^{(\text{ad})}_{i+\frac{1}{2},j,k}+\Delta\rho'^{(\text{ad})}_{i-\frac{1}{2},j,k}} + \left(q'_{1\,i,j,k} + 2q'_{2\,i,j,k}z_{i,j,k} + \ldots.\right)\left.\frac{\partial z}{\partial\xi}\right|_{i,j,k},$$

the above guarantees monotonic stratification of cubic polynomial interpolant for density critical for PGF static stability; simple differencing does not

Boussinesq approximation and stiffening of EOS

Dukowicz, 2001 idea [also Sun, Bleck, Rooth, Dukowicz, Chassignet, & Killworth, 1999]: most variation of *in situ* density occur due to changes in pressure, and a much smaller fraction due to changes in Θ, S, hence

$$\rho = r(P) \cdot \rho^{\bullet}(\Theta, S, P)$$

where $r(P)$ is a universal function (*does not depend on local* Θ, S) which can be chosen to "absorb" most of variation of density due to pressure.

- $\rho^{\bullet}(\Theta, S, P)$ fully retains thermobaric and cabbeling effects.

- variation of Θ and S decrease with depth (survey of Levitus data)

- allows a self-consistent ($\Rightarrow$ more accurate) remapping $r(P) \to r(z)$ and $\rho^{\bullet}(\Theta, S, P) \to \rho^{\bullet}(\Theta, S, z)$ as an alternative to bulk pressure $P = g\rho_0|z|$ inside EOS

- substitution of $r(P)\cdot\rho^{\bullet}(\Theta, S, P)$ into non-Boussinesq equations shows tendency of $r(P)$ to cancel out (exactly or approximately) in all terms which depend on density: e.g., it removes spurious barotropic shear; $r(z)$ commutes with density Jacobian operator,

  $$\mathcal{J}_{x,s}(r(z)\cdot\rho^{\bullet}, z) = r(z)\cdot\mathcal{J}_{x,s}(\rho^{\bullet}, z)\,,$$

  and BVF computed using a Boussinesq-like rule

  $$N^2 = -\frac{q}{\rho_0^{\bullet}}\left[\left.\frac{\partial\rho^{\bullet}}{\partial\Theta}\right|_{S,z=\mathrm{const}}\frac{\partial\Theta}{\partial z} + \left.\frac{\partial\rho^{\bullet}}{\partial S}\right|_{\Theta,z=\mathrm{const}}\frac{\partial S}{\partial z}\right]$$

  is closer to its non-Boussinesq counterpart than in the case of standard Boussinesq approximation [$\rho_0^{\bullet}$ is a constant similar to Boussinesq reference density, but representing $\rho^{\bullet}$ instead of *in situ* density]

Practical "stiffened" EOS for ROMS

From EOS of Jackett & McDougall, 1995,

$$\rho(\Theta,S,z)=\frac{\rho_1(\Theta,S)}{1-0.1\cdot z\,/[K_{00}+K_0(\Theta,S)+K_1(\Theta,S)\cdot z+K_2(T,S)\cdot z^2]}$$

chose

$$r(z)=\frac{1}{1-0.1z/K^{\text{ref}}(z)}\quad\text{with}\quad K^{\text{ref}}(z)=K_{00}+K_0^{\text{ref}}+K_1^{\text{ref}}z+K_2^{\text{ref}}z^2$$

and $K_0^{\text{ref}}=K_0\left(\Theta^{\text{ref}},S^{\text{ref}}\right)$, $K_1^{\text{ref}}=K_1\left(\Theta^{\text{ref}},S^{\text{ref}}\right)$, $K_2^{\text{ref}}=K_2\left(\Theta^{\text{ref}},S^{\text{ref}}\right)$.

Select representative *abyssal* values $\Theta^{\text{ref}}=3.5$, $S=34.5$, then $K_0^{\text{ref}}=2924.921$, $K_1^{\text{ref}}=0.34846939$, $K_2^{\text{ref}}=0.145612\times10^{-5}$, and $\rho_1\left(\Theta^{\text{ref}},S^{\text{ref}}\right)=1027.43879$.

The "stiffened" EOS becomes

$$\rho^{\bullet\prime}(\Theta,S,z)=[\rho_0^\bullet+\rho_1'(\Theta,S)]\cdot\frac{1-0.1z/K^{\text{ref}}(z)}{1-0.1z/K(\Theta,S,z)}-\rho_0^\bullet$$

$\rho^{\bullet\prime}$ is perturbation of $\rho^\bullet$ relatively to a constant reference value $\rho_0^\bullet$, for which $\rho_0^\bullet=\rho_1\left(\Theta^{\text{ref}},S^{\text{ref}}\right)$ is the natural choice. Cancellation of large terms yields

$$\begin{aligned}\rho^{\bullet\prime}(\Theta,S,z)&=\rho_1'(\Theta,S)+0.1z\cdot\frac{\rho_0^\bullet+\rho_1'(\Theta,S)}{K_{00}+K_0^{\text{ref}}+K_1^{\text{ref}}z+K_2^{\text{ref}}z^2}\times\\&\times\quad\frac{K_0^{\text{ref}}-K_0(\Theta,S)+\left(K_1^{\text{ref}}-K_1(\Theta,S)\right)\cdot z+\left(K_2^{\text{ref}}-K_2(\Theta,S)\right)\cdot z^2}{K_{00}+K_0(\Theta,S)+(K_1(\Theta,S)-0.1)\,z+K_2(\Theta,S)z^2}\\&=\rho_1'(\Theta,S)+\widetilde{q}'(\Theta,S,z)\cdot z\end{aligned}$$

so far without any approximation.

Property $\rho^{\bullet\prime}\left(\Theta^{\text{ref}}, S^{\text{ref}}, z\right) \equiv 0$, and, similarly, $\widehat{q}'\left(\Theta^{\text{ref}}, S^{\text{ref}}, z\right) \equiv 0$ regardless of z ensures that variation of $\rho^{\bullet\prime}$ is expected to be small, and decrease with depth because variation of Θ and S also decrease.

Already close to the desired form, but $\widehat{q}'(\Theta, S, z)$ still explicitly depends on z, although the dependency is weak in comparison with the original JM95. Taylor expansion for powers of z yields

$$\rho^{\bullet\prime}(\Theta, S, z) = \rho_1'(\Theta, S) + q_1'(\Theta, S) \cdot z$$

where

$$q_1'(\Theta, S) = 0.1 \cdot \left[\rho_0 + \rho_1'(\Theta, S)\right] \cdot \frac{K_0^{\text{ref}} - K_0(\Theta, S)}{\left[(K_{00} + K_0(\Theta, S)) \cdot \left(K_{00} + K_0^{\text{ref}}\right)\right]}$$

with q_1' does not depend on z.

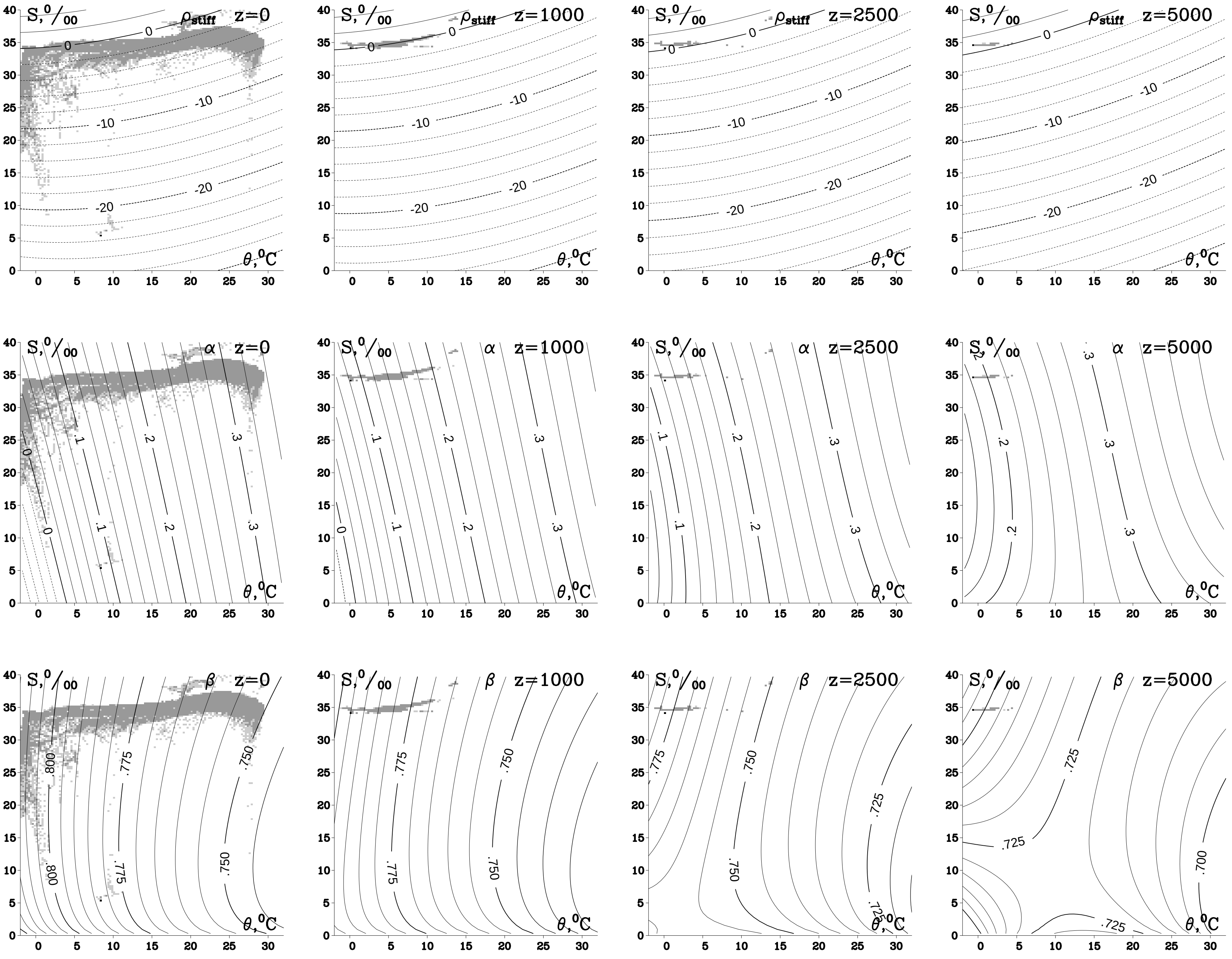
This involves **an approximation**, and discards all coefficients associated with K_1 and K_2 terms (hence 14 out of 26 in the original JM95 bulk secant modulus). Naturally, this raises concern about the accuracy.

The final practical version is,

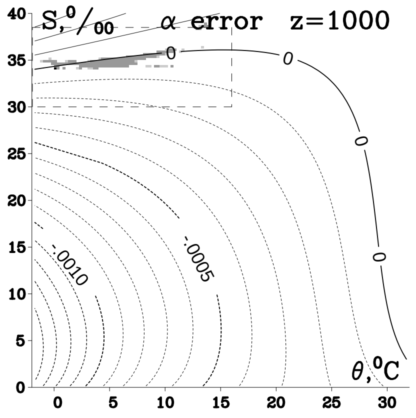
$$\rho^{\bullet\prime}(\Theta, S, z) = \rho_1'(\Theta, S) + q_1'(\Theta, S) \cdot z\,(1 - \gamma z)$$

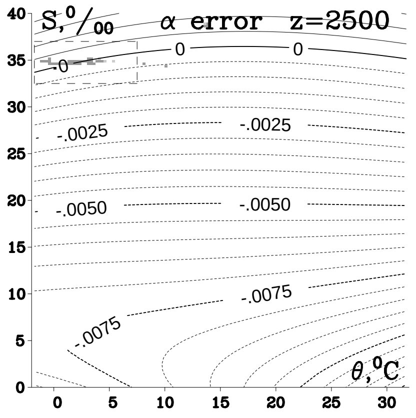
with $\gamma = 1.72 \times 10^{-5}$ for Θ^{ref} and S^{ref} values selected above (γ is just a constant).

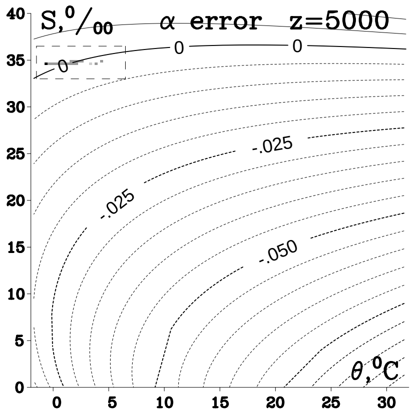
Practical "stiffened" EOS: Continued

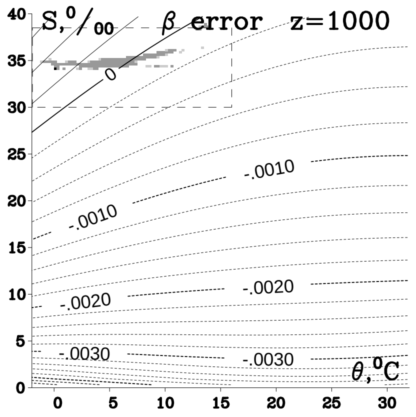


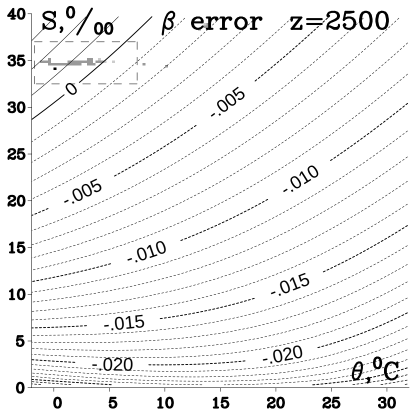
Practical "stiffened" EOS: Assessing accuracy

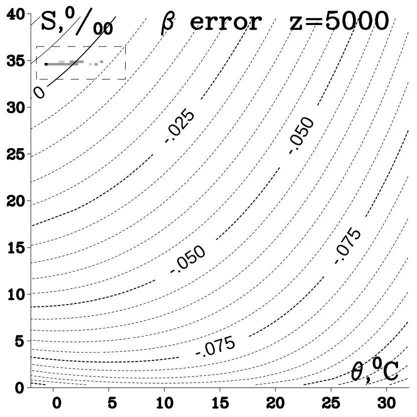


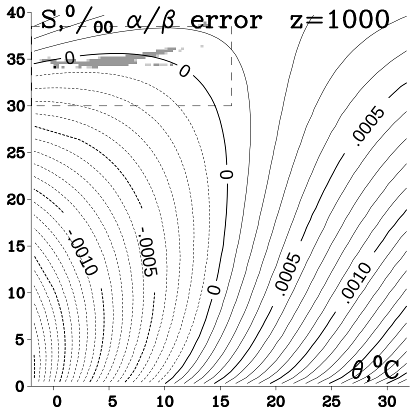


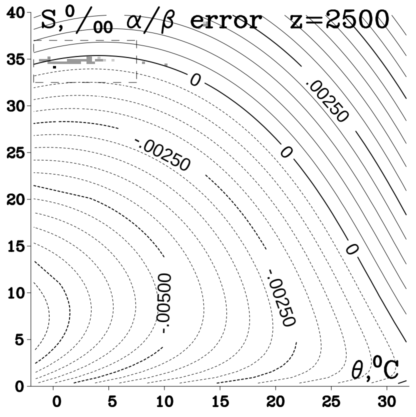


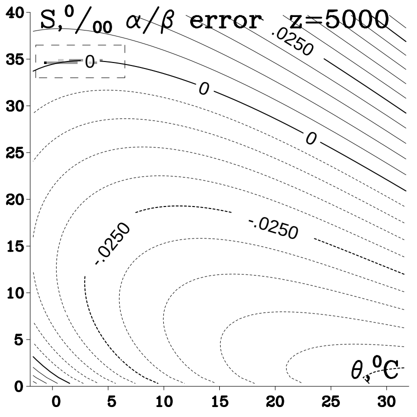












Summary of EOS

- Follows Dukowicz, 2001, except in choosing constant Θ, S reference to construct $r(P) \rightarrow r(z)$, rather than globally averaged profile from Levitus. This is to facilitate adiabatic differencing critical for PGF in ROMS (z-coordinate models do not care)
- If Boussinesq approximation is applied, **it must be applied to EOS as well**
- in comparison with 2003 PGF study allows to align zero-error point on Θ, S plane with the desired location. Approximately one order of magnitude more accurate.
- fully retains thermobaric and cabbeling effects
- adiabatic derivatives normalized by stiffened reference density, e.g.,

$$N^2 = -\frac{q}{\rho_0^\bullet}\left[\left.\frac{\partial\rho^\bullet}{\partial\Theta}\right|_{S,z=\text{const}}\frac{\partial\Theta}{\partial z} + \left.\frac{\partial\rho^\bullet}{\partial S}\right|_{\Theta,z=\text{const}}\frac{\partial S}{\partial z}\right]$$

  are close to that from non-Boussinesq model
- removes most (up to $\sim 90\%$) of Boussinesq approximation errors; replaces $\rho_0 = const$ reference with $\rho_0^\bullet r(z)$, which closer to reality
- the reason why it works well is because $r(z)$ in *integrable*, merely because $r \sim e^{gz/c^2} \sim 1 + gz/c^2$, with $gz/c^2 << 1$ so both density and pressure are multiplied by approximately the same factor (exactly the same in barotropic case), resulting in cancellation $r(z)$ (approx. or exact), and preserving semantics of Boussinesq code
- Eliminates mode splitting error in computing ρ^*, $\overline{\rho}$ without increase of code complexity (basically without any change in parts computing ρ^*, $\overline{\rho}$): now these two are purely baroclinic (no spurious stratification), and therefore assumption that they are kept constant during fast-time stepping is fully justified

Flather Boundary Conditions

Basic idea: rewrite

$$\frac{\partial u}{\partial t} = -g\frac{\partial \zeta}{\partial x} \qquad\qquad \frac{\partial \zeta}{\partial t} = -h\frac{\partial u}{\partial x}$$

in terms of *characteristic variables* (Riemann invariants)

$$\mathcal{R}^{\pm} = u \pm \sqrt{\frac{g}{h}}\cdot\zeta \qquad \Rightarrow \begin{cases} \partial_t\mathcal{R}^+ + c\partial_x\mathcal{R}^+ = 0 \\ \\ \partial_t\mathcal{R}^- - c\partial_x\mathcal{R}^- = 0 \end{cases} \qquad\qquad c = \sqrt{gh}$$

$\mathcal{R}^-$, $\mathcal{R}_0^+$ move to left and right independently from each other, also $\mathcal{R}^- = \mathcal{R}_0^-(x{+}ct) = const$ and $\mathcal{R}_0^+ = \mathcal{R}_0^+(x - ct) = const$ along their characteristics $x \pm ct = const$.

boundary conditions are self-obvious for $\mathcal{R}^-$, $\mathcal{R}_0^+$:

left side

$$\mathcal{R}^+ = u^{(\text{ext})} + \sqrt{\frac{g}{h}}\cdot\zeta^{(\text{ext})}$$

$\mathcal{R}^-$ free radiation b.c.

right side

$\mathcal{R}^+$ free radiation b.c.

$$\mathcal{R}^- = u^{(\text{ext})} - \sqrt{\frac{g}{h}}\cdot\zeta^{(\text{ext})}$$

then transform $\mathcal{R}^+$,$\mathcal{R}^-$ back to the original variables u, ζ

What is missing above is the fact that on a staggered grid u, ζ are not co-located, which obscures $u,\zeta \leftrightarrow \mathcal{R}^+,\mathcal{R}^-$ translation: $\Rightarrow$ *ad hoc* interpolations $\Rightarrow$, reflections, instability, restriction on permissible time step; excessive tidal amplitudes. ...Riemann solvers are known for 40 years, they use non-staggered grids.

Approach:

- radiate out (next slide for details) u, ζ *independently* from the other to a common location at new time step,

$$\to u^* = \tilde{u}^{n+1}_{j+1/2} \qquad\qquad \to \zeta^* = \tilde{\zeta}^{n+1}_{j+1/2}$$

- construct *outgoing* characteristic variable, $\mathcal{R}^+$ or $\mathcal{R}^-$, using u^*, ζ^*:
  say, on the right-side boundary, assembly

$$\mathcal{R}^+ = u^* + \sqrt{\frac{g}{h}} \cdot \zeta^*$$

  prescribe

$$\mathcal{R}^- = \left(\mathcal{R}^-\right)^{(\text{ext})} = u^{(\text{ext})} - \sqrt{\frac{g}{h}} \cdot \zeta^{(\text{ext})}$$

- translate back

$$u^{n+1}_{j+1/2} = \frac{\mathcal{R}^+ + \mathcal{R}^-}{2} = \frac{u^* + u^{(\text{ext})}}{2} + \sqrt{\frac{g}{h}} \cdot \frac{\zeta^* - \zeta^{(\text{ext})}}{2}$$

This is **different** from the original Flather condition $u = u^{(\text{ext})} + \sqrt{g/h}\left(\zeta^* - \zeta^{(\text{ext})}\right)$

Overall follows Blayo and Debreu, 2004, **except the "radiate out" step**

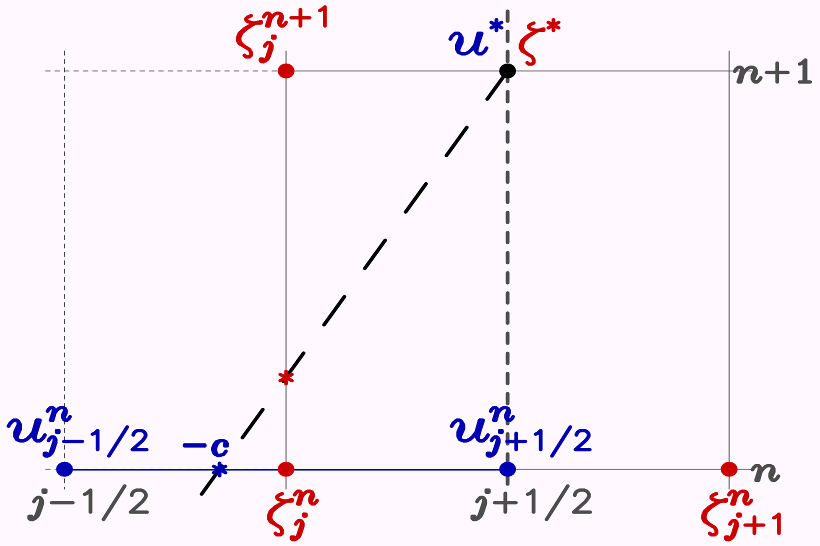
$\mathcal{R}^-$ and $\mathcal{R}^-$ never appear explicitly in the code

boundary conditions for ζ (i.e., setting ζ at ghost points half-grid outside the boundary row of u-points) are needed only by the radiation scheme, but are not needed outside the Flather B.C. The are therefore auxiliary. Use explicit radiation scheme.

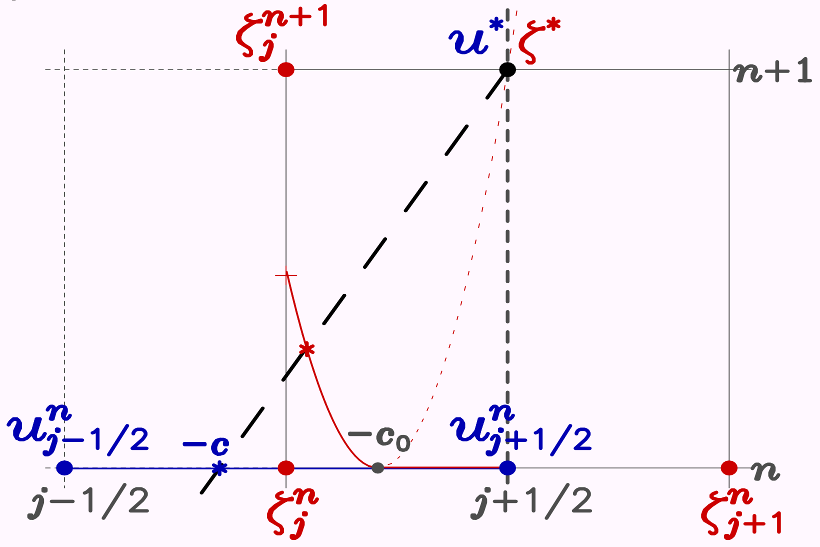
Flather algorithm for staggered grid

explicit radiation scheme for u,

$$u^* = (1-c)u^n_{j+1/2} + cu^n_{j-1/2}$$



.



explicit-implicit switch for ζ:

$$\zeta^* = \tilde{\zeta}^{n+1}_{j+1/2} = \zeta^n_j\left(\frac{1}{2}+c\right) + \zeta^n_{j+1}\left(\frac{1}{2}-c\right)$$

if $c < 1/2$; and

$$\zeta^* = \frac{\zeta_j^n + \zeta_j^{n+1}(2c-1)}{2c}$$

if $c > 1/2$; relies on auxiliary B.C.

$$\zeta_j^{n+1} = (1-c)\zeta^n_{j+1} + c\zeta^n_j$$

unstable, hole around $c \approx 1/2$

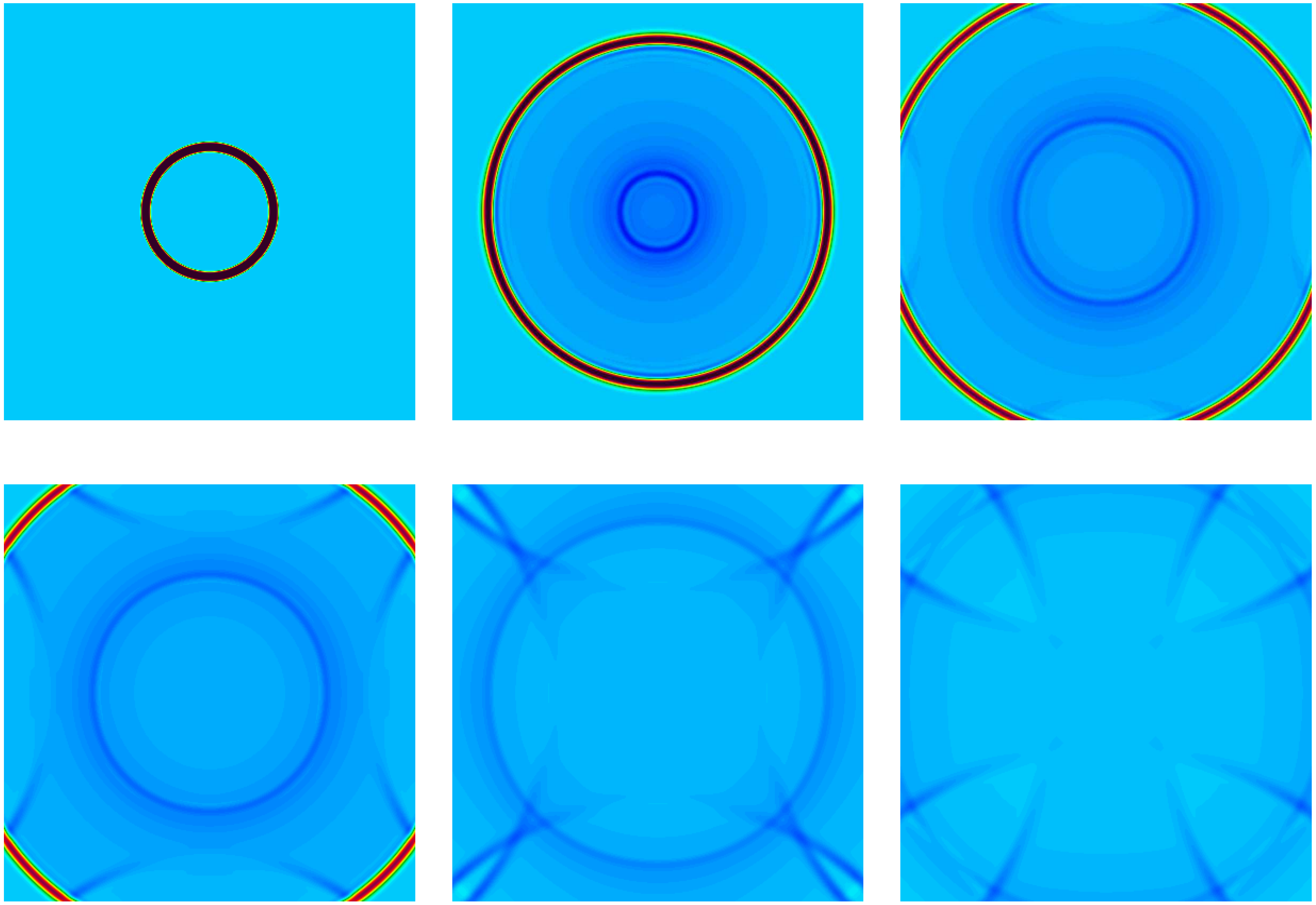
Avoid single-point value $\zeta^n_{j-1/2}$ if $c = 1/2$

$$\zeta^* = \zeta^n_j\left(\frac{1}{2}+c\right) + \zeta^n_{j+1}\left(\frac{1}{2}-c\right)$$

if $c < c_0$; and

$$\zeta^* = \zeta_j^n\left[\frac{1}{2} + c_0\left(2-\frac{c_0}{c}\right) - \left(1-\frac{c_0}{c}\right)^2\right]$$
$$+\zeta^n_{j+1}\left[\frac{1}{2} - c_0\left(2-\frac{c_0}{c}\right)\right]$$
$$+\zeta_j^{n+1}\left(1-\frac{c_0}{c}\right)^2$$

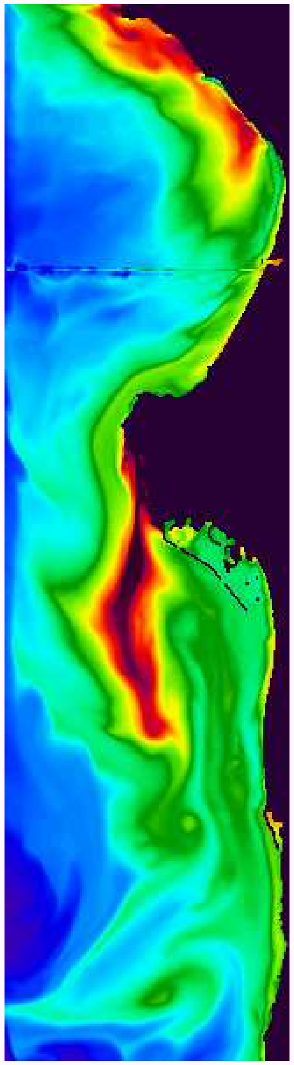
if $c > c_0$; $c_0 = 1/\left(2+\sqrt{2}\right)$; **stable**;



Flather algorithm summary

- replaces Flather, 1976; Chapman, 1985 algorithms
- original setting $\zeta^* = \left(\zeta_j^n + \zeta_{j+1}^n\right)/2$ becomes unstable once $c > 1/2$. This is about 40% more restrictive than the native stability limit of Generalized FB barotropic mode, c=0.87
- $\zeta^* = \left(\zeta_j^{n+1} + \zeta_{j+1}^n\right)/2$ removes the restriction, but both of them are more reflective than the switched explicit-implicit version
- level of wave reflection of the final algorithm is not achievable by an Orlanskii scheme. A fraction oF percent
- the **hole** instability, $\sim 0.48 < c <\sim 0.52$, was first encountered by Xavier Capet (shows up as blow-up at the boundary with details highly dependent on setting of time step, incl. no blow-up, when $c < 1/2$), and went unexplained for several months. Now we are able to reproduce it in idealized wave problem
- the switched explicit-implicit algorithm for ζ^* was ported into AGRIF (along with the barotropic mode as a whole), but the final version is not
- we can do tides using S-shaped fast-time averaging without instabilities or excessive amplitudes
- out of 30+ publications related to the subject, only Nycander & Döös, 2004 analyze effects of discretisation and time stepping

Thread programming and OpenMP Standard



ROMS OpenMP code was originally written under assumption "write-back" *cache coherency protocol* designed by SGI for its PowerChallenge and Origin 2000 supercomputers. The protocol keeps track on multiple copies of data stored in the same `cache_line` in main memory, which reside in different caches on an SMP machine. Once a cached copy of the `cache_line` is modified by one of the CPUs, it is marked as "exclusive" for that CPU; "dirty" in main memory, and "invalid" in all other caches. Processor is not allowed (by hardware; 2-bits per `cache_line` – the only data which travels directly from CPU to CPU) to work on "invalid" or to read "dirty" `cache_line`, forcing the owner of "exclusive" copy to write it back into the main memory when it is requested by another CPU.

With the departure of Origin 2000 the above protocol is no longer honored.

OpenMP standard specifies that a shared variable is flushed automatically at the end of parallel region, barrier, lock *only if that variable is visible* from the programming unit where that OpenMP directive is present. **ROMS (all codes) did not follow this rule. The problem is fixable (fixed), but requires attention**

OpenMP Version 2.5 Specification is released on May 2005; updates v.2.1 of November 2001. More explicit about F90 (e.g., explains insidents of *false sharing* due to compiler-inserted data copying).

??? Intel, AMD are very vague about their cache-coherency protocols

??? looks like Intel 9.1.x compiler ignores !$OMP FLUSH

Left: an example of loss of synchronization; field is near-surface salinity; there are 64 subdomains and 4 threads; thread #3 (the upper) is out of sync. These kind off errors are non-deterministic, non-reproducible, and extremely hard to debug.

ROMS Linux cluster computing: Experiences

- Opportunity to harness supercomputing power at manageable cost

- Scaling is not an issue for ROMS-family codes: adding more nodes leads to nearly proportional increase of computing speed, even if using GigE interconnect for a do-it yourself cluster size ($\leq$ 24 nodes)

- ROMS code in OpenMP mode optimally tiled for cache utilization outperforms 2:1 ROMS code in MPI mode *within* a single SMP node. Same observation applies for single processor, tiled vs. single block. However, OpenMP is applicable for one node only

- A common major disappointment with GigE clusters is that CPU usage (as seen by `top`) is well below 100%: typically $\sim$50% or less.

- The two items above ($\sim$50% CPU utilization $\times$ 1/2 due poor cache utilization) bring us to $\sim$25% efficiency relatively to the **dream code** which combines OpenMP-level single-processor performance with MPI scaling on multiple nodes

Linux cluster computing continued..

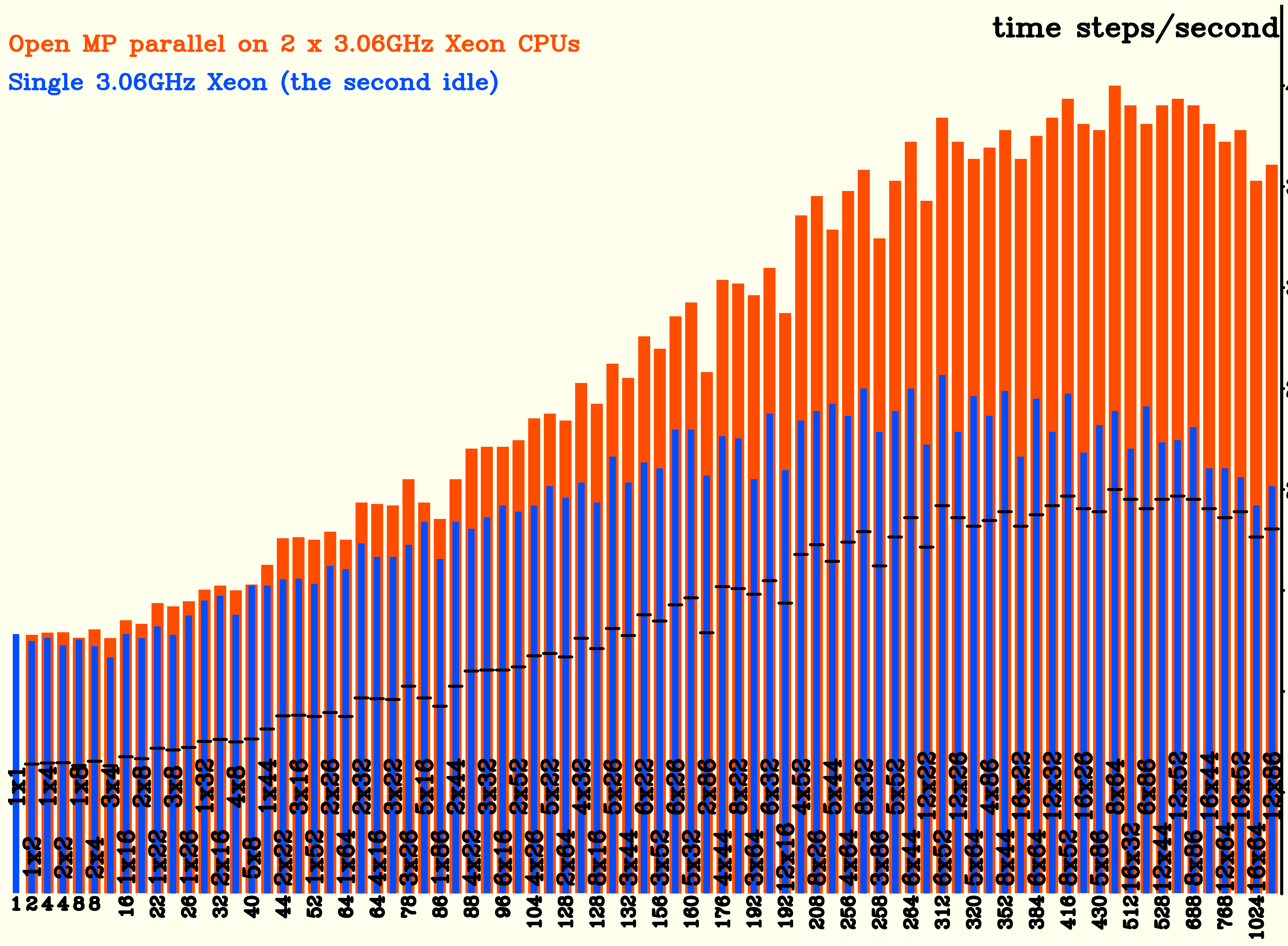
- most today's Linux clusters are made of 2- 4-way SMP nodes
  - dual CPU; dual- or quad-core CPU
  - shared memory buses
  - sometimes shared caches
  - multiple/shared NIC/interconnect cards
  - GigE is full duplex, however full-duplex transmission can be used *for point-to-point connections* only (e.g., `node1` $\leftrightarrow$ `node2` is full duplex, but if `node1` $\longrightarrow$ `node2` and `node2` $\longrightarrow$ `node3`, then `node2` in not in full-duplex mode; intelligent switches can alleviate this)

- most MPI codes are designed to
  - separate communications from computations in time resulting in peak loads on interconnects followed periods of by inactivity
  - treat multiple processors (cores) within each physical node as separate compute nodes, ignoring non-uniform topology of the machine, (if any).
  - introduce competition for shared interconnect cards
  - single subdomain — single processor policy
  - ignore cache effects

In ROMS community we started as multi-threaded (Cray, SGI, then OpenMP) code designed for SMP. Then, after the departure SGI Origin we *de-facto* downplayed importance of SMP computing as limited to single node: MPI rules.

- Now SMPs are coming back in form of multicore CPUs. 2.4GHz Q6600 Intel Core 2 **Quad** is only $270 at `mwave.com` $\Rightarrow$ era of single-processor computing is over

Cache effects on SMP computer



two-dimensional Soliton problem on 768×256 grid

Basic stuff: matched sends and receives

```
node1                      node2
MPI_Send (..,node2,..)     MPI_Recv (..,node1,..)
MPI_Recv (..,node2,..)     MPI_Send (..,node1,..)
```

- most recommended in textbooks
- safe against deadlock regardless of size of system buffer
- synchronous communications allows replacement `MPI_Send` → `MPI_Ssend`
- low latency
- if hardware allows **full-duplex** communication, this capability is not utilized

```
node1                                  node2
MPI_SendRecv[_Replace] (..,node2,..)   MPI_SendRecv[_Replace] (..,node1,..)
```

- recommended by IBM "Red Books"
- guarantees no deadlock
- leaves everything up to vendor of particular MPI implementation, and hopes that they do it the most optimal way

```
node1                          node2
MPI_Irecv (..,node2,req,..)    MPI_Irecv (..,node1,req,..)
MPI_Send (..,node2,..)         MPI_Send (..,node1,..)
MPI_Wait (req,status)          MPI_Wait (req,status)
```

- most halo exchanges are done this way
- utilized **full-duplex** communication ...as long as nobody else interferes

Halo-exchange algorithms

Generally considered a widely studied problem, Alan Wallcraft, but not trivial, even after many years

- Communication with more than one neighbor: scheduling of multiple messages
- Revisiting with emphasis for SMP hardware nodes: dealing with interferences of MPI-nodes running on the same board and sharing network interface
- Hardware caveats: can we make use of second NIC-card interface on the board?

stage 1: north-south

```
do iter=0,1
  if (mod(jnode+iter,2)==0) then
    MPI_SendRecv (..,south,...)
  else
    MPI_SendRecv (..,north,..)
  endif
enddo
```

stage 2: east-west, inc. corners

```
do iter=0,1
  if (mod(inode+iter,2)==0) then
    MPI_SendRecv (..,west,..)
  else
    MPI_SendRecv (..,east,..)
  endif
enddo
```

- two-stage algorithm: sending east-west messages must wait until north-south are received because corner points should be included into outgoing messages. two-stage $\Rightarrow$ added latency
- no need to worry about `Send`—`Recv` message matching, however order of which, north- or southbound, message should be send first is alternated for adjacent nodes: this is what `if (mod(jnode+iter,2)==0) then` is for
- for performance relies exclusively on MPI-vendor implementation
- guarantees no deadlock (MPI standard)
- full-duplex transmission is possible if implemented by vendor inside `MPI_SendRecv`
- presence of a second MPI-node within the same physical SMP node which tries to send messages to a different destination through the same NIC destroys point-to-point arrangement, ultimately loosing full-duplex transmission

alternating `Send-Recv` **—** `Recv-Send` **order, separately in each direction**

stage 1: north-south

```
  do iter=0,1
    if (mod(jnode+iter,2)==0) then
      MPI_Ssend (..,south,..)
      MPI_Recv (..,south,..)
    else
      MPI_Recv (..,north,..)
      MPI_Ssend (..,north,..)
    endif
  enddo
```

stage 2: east-west, inc. corners

```
  do iter=0,1
    if (mod(inode+iter,2)==0) then
      MPI_Ssend (..,west,..)
      MPI_Recv (..,west,..)
    else
      MPI_Recv (..,east,..)
      MPI_Ssend (..,east,..)
    endif
  enddo
```

half-duplex

chequerboard algorithm

stage 1: north-south

```
  do iter=0,1
    if (mod(inode+jnode+iter,2)==0) then
      MPI_Ssend (..,south,..)
      MPI_Recv (..,south,..)
    else
      MPI_Recv (..,north,..)
      MPI_Ssend (..,north,..)
    endif
  enddo
```

stage 2: east-west, inc. corners

```
  do iter=0,1
    if (mod(inode+jnode+iter,2)==0) then
      MPI_Ssend (.,west,..)
      MPI_Recv (..,west,..)
    else
      MPI_Recv (..,east,..)
      MPI_Ssend (..,east,..)
    endif
  enddo
```

fully matched send-receives:
"white" nodes do `Send` first, then `Recv`;
"black"s do the opposite;

if two adjacent MPI-subdomains are running on the same SMP node and share network interface, then when one MPI-node sends message to the south, approximately at the same time the other one receives from the north, and vice versa. Neither of these algorithms takes advantage of full-duplex transmission.

4-color chequerboard algorithm

stage 1: north-south

```
do iter=0,1
  if (mod(jnode+iter,2) == 0) then
    if (mod(inode+iter,2) == 0) then
      MPI_Ssend (..,south,..)
      MPI_Recv (..,south,..)
    else
      MPI_Recv (..,south,..)
      MPI_Ssend (..,south,..)
    endif
  else
    if (mod(inode+iter,2) == 0) then
      MPI_Recv (..,north,..)
      MPI_Ssend (..,north,..)
    else
      MPI_Ssend (..,north,..)
      MPI_Recv (..,north,..)
    endif
  endif
enddo
```

stage 2: east-west, inc. corners

```
do iter=0,1
  if (mod(inode+iter,2) == 0) then
    if (mod(jnode+iter,2) == 0) then
      MPI_Ssend (..,west,..)
      MPI_Recv (..,west,..)
    else
      MPI_Recv (..,west,..)
      MPI_Ssend (..,west,..)
    endif
  else
    if (mod(jnode+iter,2) == 0) then
      MPI_Recv (..,east,..)
      MPI_Ssend (..,east,..)
    else
      MPI_Ssend (..,east,..)
      MPI_Recv (..,east,..)
    endif
  endif
enddo
```

if two MPI-subdomains running on the same physical SMP node are adjacent to each other in either east-west or north-south directions, then messages transmitted in the transversal direction by the two MPI processes communicate with the same physical destination, but travel in opposite directions: each of the two MPI-processes makes half-duplex transmissions, physical node is in *full-duplex* mode.

- **the fastest algorithm on NCSA "tungsten" cluster** (dual-Xeon nodes, Myranet interconnect); we were able to use up to 384 CPUs (192 nodes) and it scales
- the innermost "`if`"-blocks can be replaced with `MPI_SendRecv`'s (this is `mpi_exchange4SR.F`), or with `IRecv` − `Send` − `Wait` sequences. Either alternative is slower on dual-CPU nodes (?). Counterintuitive.

Single-stage, 8-message, direct exchange

```
MPI_Irecv (..,west,..)
MPI_Irecv (..,south,..)
MPI_Irecv (..,east,..)
MPI_Irecv (..,north,..)
...
MPI_Irecv (..,north-west,..)
create list of messages to be received

MPI_Send (..,west,..)
MPI_Send (..,south,..)
MPI_Send (..,east,..)
MPI_Send (..,north,..)
...
MPI_Send (..,north-west,..)

do while (until the list is empty)
  MPI_Waitany (...list, index, ...)
  unpack incoming message
  and delete it from the list
enddo
```

- asynchronous, single-stage: all messages can be send at-once without waiting for any to be received. This is `mpi_exchange8WA.F`
- designed to hide latencies of different messages behind each other
- corners must be sent separately as small messages; this is price paid for being single-stage
- explored a variant where `Send` section uses alternating chequer-board sequence to match order of sending between adjacent nodes: this has a very little effect
- use of `MPI_Waitany` rather than individual `MPI_Wait`'s results in lower CPU usage in GigE cluster, but the code runs overall faster with `Waitany` (less *busy waiting*)
- no though about interference of multiple MPI-nodes running on the same physical node: just hope that hardware can handle multiple competing sends
- overall this is **the fastest** algorithm on our own GigE cluster with dual-Opteron nodes (all CPUs are single core) and short cables connecting `eth1`'s of consecutive nodes (this is discussed below)
- **helplessly slow** on NCSA Myranet cluster

Tweaking hardware: Making use of the second NIC card

virtually all server boards come with it

computer consultants tell don't bother: it won't improve latencies or anything else, however

- ROMS has very simple, static partitioning topology, which can be easily mapped onto the machine, and
- Physical nodes are SMP nodes, and the code tends to send multiple messages at the same time, and
- MPI works by hostnames and **knows nothing** about IP addresses: you can trick it to believe that hosts are the same, while delivering message using different lines

Connect the otherwise unused `eth1`'s of pairs consecutive nodes by short cables and take advantage of dedicated lines between some of the pairs of nodes.

Connect both NIC interfaces of the head-node to the main switch and assign alternating IPs for NFS-mounting of disks. This effectively doubles the I/O bandwidth if multiple compute nodes attempt to write at the same time.

node3

ifcfg-eth0
```
DEVICE=eth0
IPADDR=10.1.1.3
NETMASK=255.255.255.0
```

ifcfg-eth1
```
DEVICE=eth1
IPADDR=192.168.1.3
NETMASK=255.255.255.0
```

/etc/hosts
```
...
10.1.1.2      node2
10.1.1.3      node3
192.168.1.4   node4
10.1.1.5      node5
10.1.1.6      node6
10.1.1.7      node7
...
```

node4

ifcfg-eth0
```
DEVICE=eth0
IPADDR=10.1.1.4
NETMASK=255.255.255.0
```

ifcfg-eth1
```
DEVICE=eth1
IPADDR=192.168.1.4
NETMASK=255.255.255.0
```

/etc/hosts
```
...
10.1.1.2      node2
192.168.1.3   node3
10.1.1.4      node4
10.1.1.5      node5
10.1.1.6      node6
10.1.1.7      node7
...
```

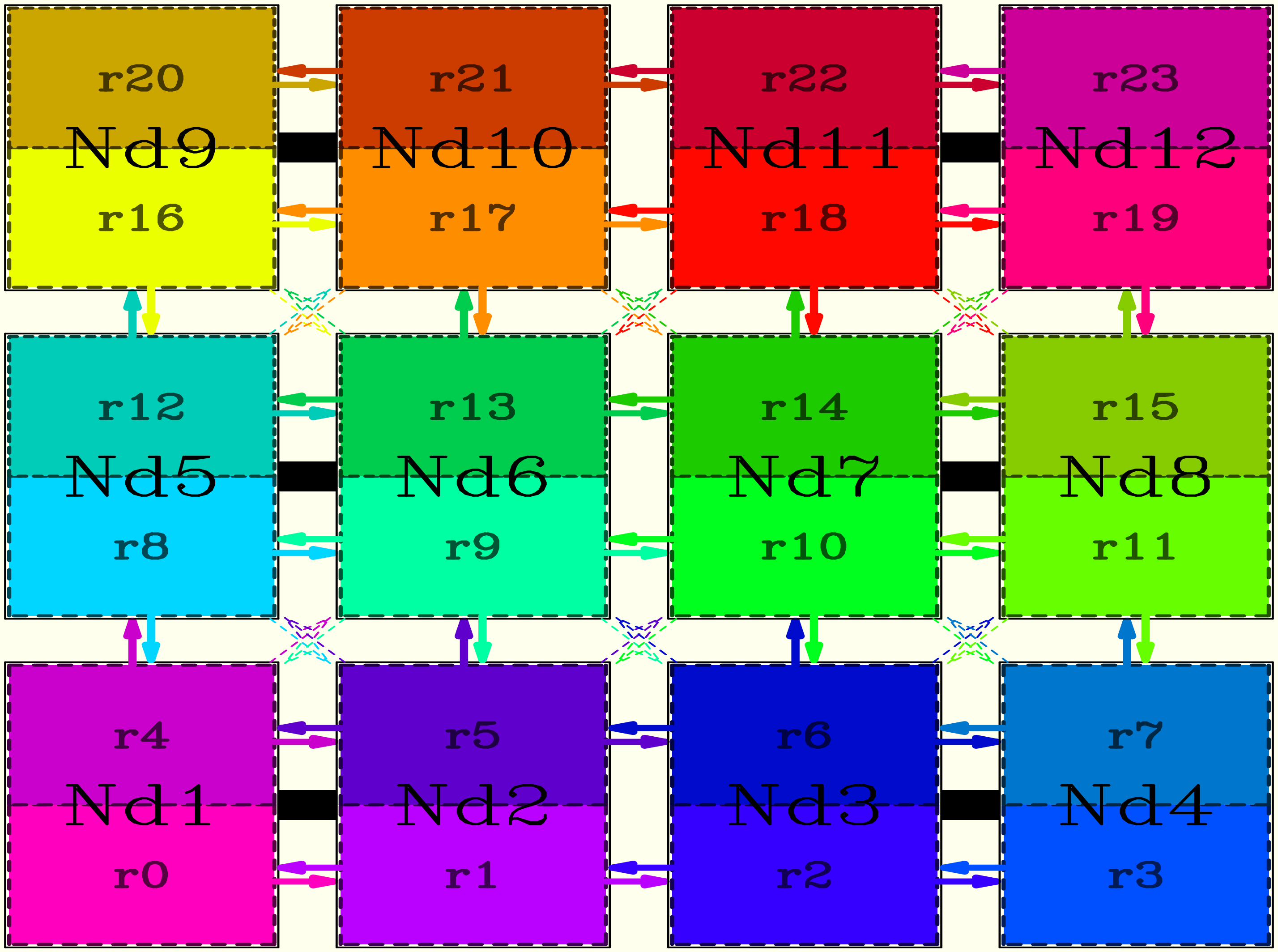
`eth0` of each node is connected to main switch

`eth1`'s of node3 and node4 are connected to each other (technically this is a separate network)

now **node4** replies to ping from **node3** as 192.168.1.4 rather than its regular IP. Similar applies to **node3** as seen from **node4**. Any MPI communication between them goes through short cable **by-passing main switch** and without competing with other messages.

use `machines.LINUX` to control node placement and maximize the use of dedicated lines

MPI-only code on Linux cluster with dual-CPU nodes

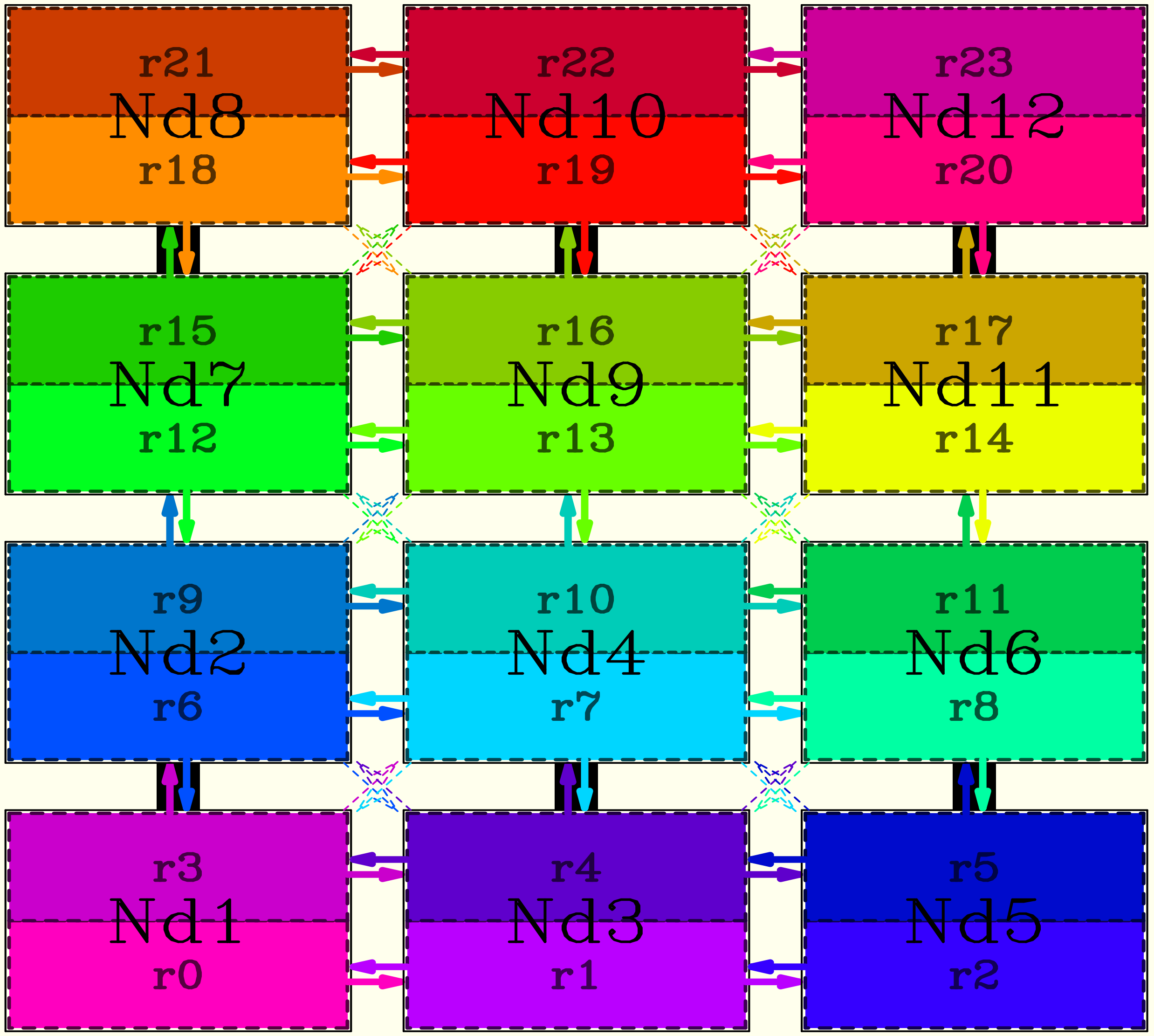


machines .LINUX

```
node1
node2
node3
node4
node1
node2
node3
node4
node8
node7
node6
node8
node8
node7
node6
node8
node9
node10
node11
node12
node9
node10
node11
node12
```

Nd1 ... Nd12 are physical compute nodes; **r0, r1, ..., r23** are MPI-ranks; `eth0`'s of each physical node are connected to switch; `eth1`'s of consecutive pairs of nodes are connected directly to each other (thick **black** lines); placement of MPI-nodes is controlled via **machines.LINUX** on the right

Another example of topology for MPI code on cluster with dual-CPU-nodes



machines .LINUX

```
node1
node3
node5
node1
node3
node5
node2
node4
node6
node2
node4
node6
node7
node9
node11
node7
node9
node11
node8
node10
node12
node8
node10
node12
```

Nd1 ... Nd12 are physical compute nodes; **r0, r1, ..., r23** are MPI-ranks; vertical **black** lines are short cables; 4 of the 7 north-south messages are within the boards; 2 via short cables, and only 1 goes through the switch

Summary of MPI-only code on cluster with dual-CPU nodes

- utilize the second NIC card on each board
- topology matters; always map your problem onto the machine
- for the Gigabit interconnect the best exchange algorithm is `MPI_Irecv` (all 8 messages, incl. corners at once) $\rightarrow$ `MPI_Send` (all) $\rightarrow$ `MPI_Wait_any` and unpack in order of appearance;
- the above exchange algorithm should not to be used on Myranet cluster; instead synchronous paired, 2-stage (east-west, then north-south; no-corners) strategy is the best there;
- contrary textbook recommendations, paired `MPI_Ssend` $-$ `MPI_Recv`; `MPI_Recv` $-$ `MPI_Ssend` works better than `MPI_SendRecv`, even on IBM (p690 at NCSA tested)
- `MPI_Srecv` exists only in PowerPoint presentations and lectures of computer science professors: google finds it, but `nm libmpi.a` does not
- **the combined effect** of topology mapping, optimized halo-exchange algorithm, and using second NIC cards yields nearly double performance (a factor of 1.8 observed) on dual-CPU node Gigabit cluster relatively to "flat MPI" approach (treating all CPUs as separate nodes); $\sim 50\% \rightarrow 90\%$ of CPU utilization observed, effectively eliminating temptation to use a more expensive Infiniband Network
- MPI-only codes tend to separate computation and communication in time resulting in peak-loads on network, while keeping it inactive during computing
- multiple CPUs running on the same SMP node compete to get usage of shared NIC cards
- scaling is near-perfect, but per-CPU performance lags behind of that of OpenMP code due to poor cache management

MPI with Threads

Now, at last, multiple threads are **officially included** into MPI-2 Standard

Replaces `MPI_Init` with `MPI_Init_thread (requested, provided)`, where

$$\texttt{requested}, \texttt{provided} = \begin{cases} 0 = \texttt{MPI\_THREAD\_SINGLE} \text{ means no thread support} \\ 1 = \texttt{MPI\_THREAD\_FUNNELED} \text{ means threads are allowed,} \\ \qquad \text{but only master thread can execute MPI calls} \\ \\ 2 = \texttt{MPI\_THREAD\_SERIALIZED} \text{ multiple threads can do} \\ \qquad \text{MPI calls, but the calls are serialized} \\ \\ 3 = \texttt{MPI\_THREAD\_MULTIPLE} \text{ means multiple threads can} \\ \qquad \text{execute concurent MPI calls} \end{cases}$$

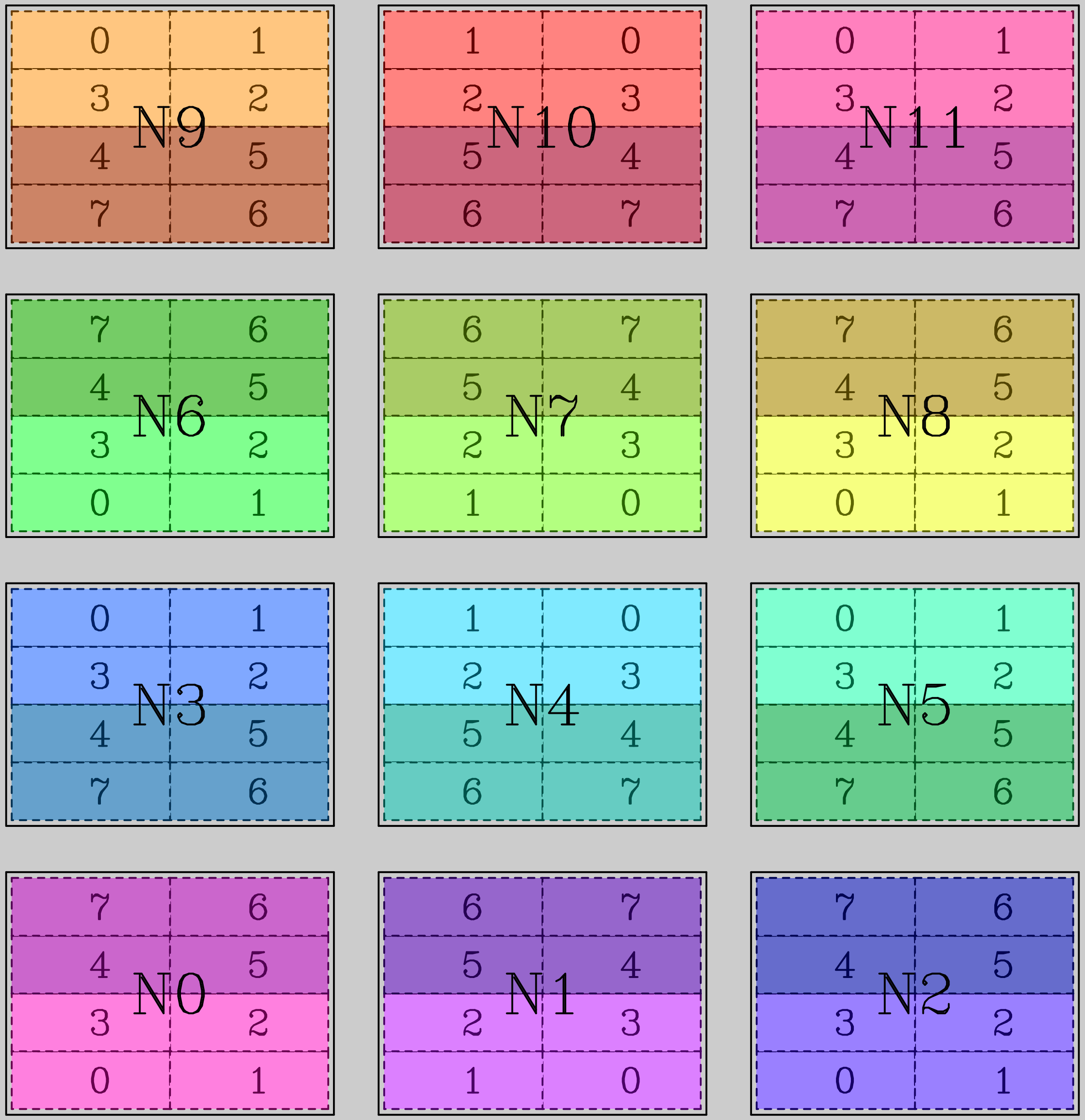
- The original motivation is to allow multiple subdomains for cache management to recover cache efficiency of optimally tiled code.
- But it turns out that there is more in it: tiling and threads can be used for tuning of scheduling of messages to alleviate competition for access to Network Interface within SMP nodes, simply put, when one thread sends messages, the other(s) compute and, and vice versa.
- Long overdue: we were talking about it for years, but it is hard to do: very careful scheduling required

Approach:

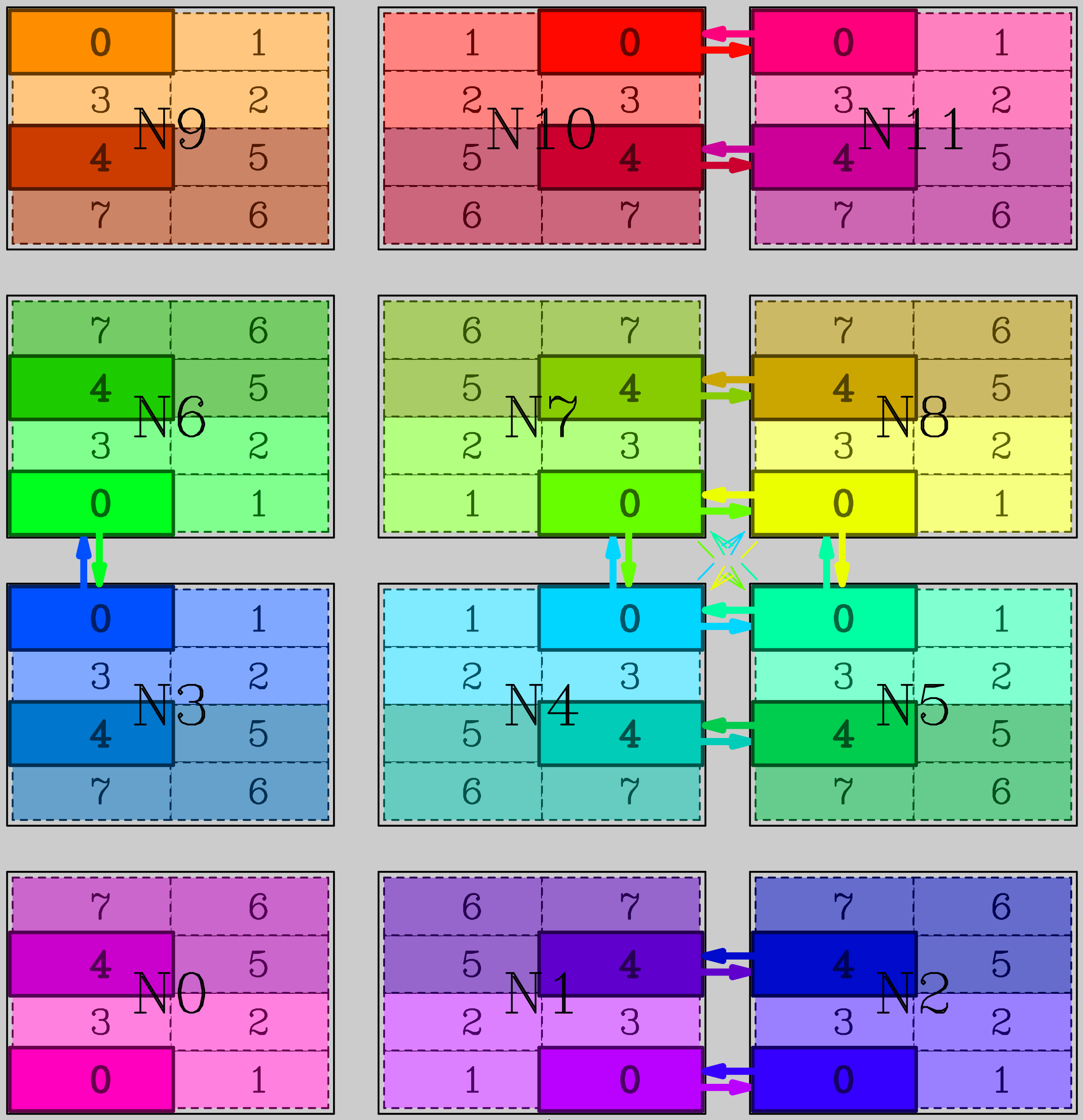
- relies on highest level of MPI thread support, `requested, provided` $=$ 3,3
- 2-level 2-dimensional subdomain decomposition: tiles within MPI subdomains;
- Once a thread completes working on a tile, it sends/receives relevant messages to its MPI-neighbor (if any). Because now there is (may be) more than one neighbor of each side, and because MPI *knows nothing* about threads (i.e., thread receiving an MPI message from MPI neighbor does not know from which thread on that node the message is coming) use **unique tags** to label messages sent by different threads;
- **Mirror** thread trajectories for adjacent MPI subdomains: that is the key to avoid deadlocks.

And, finally, the code is just a tool; **art is in its usage:** the number of possible permutations is now too large to be quickly explored.

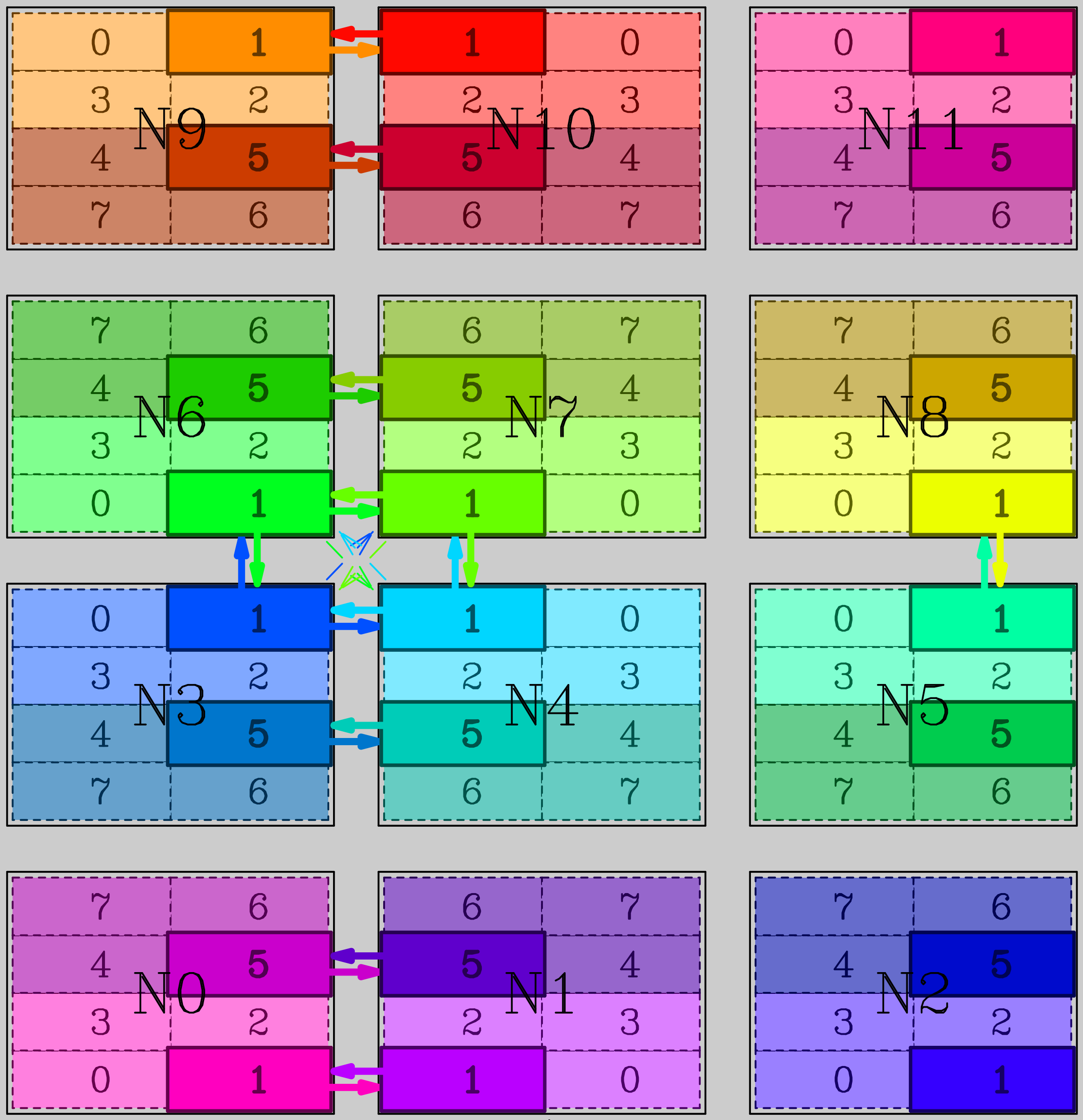
12 MPI nodes, 2 threads, 8 tiles within each node



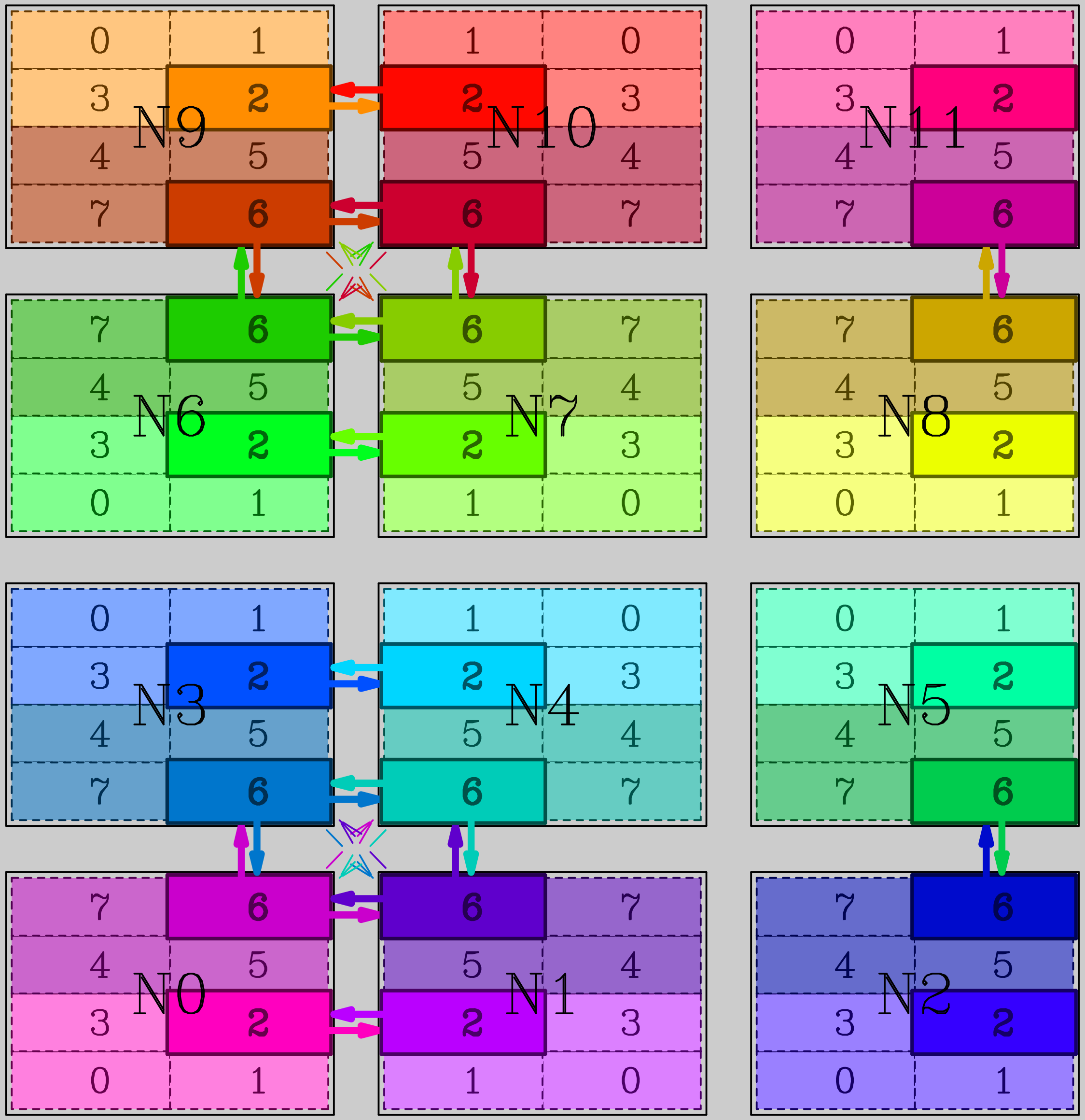
12 MPI nodes, 8 tiles, 2 threads, continued...



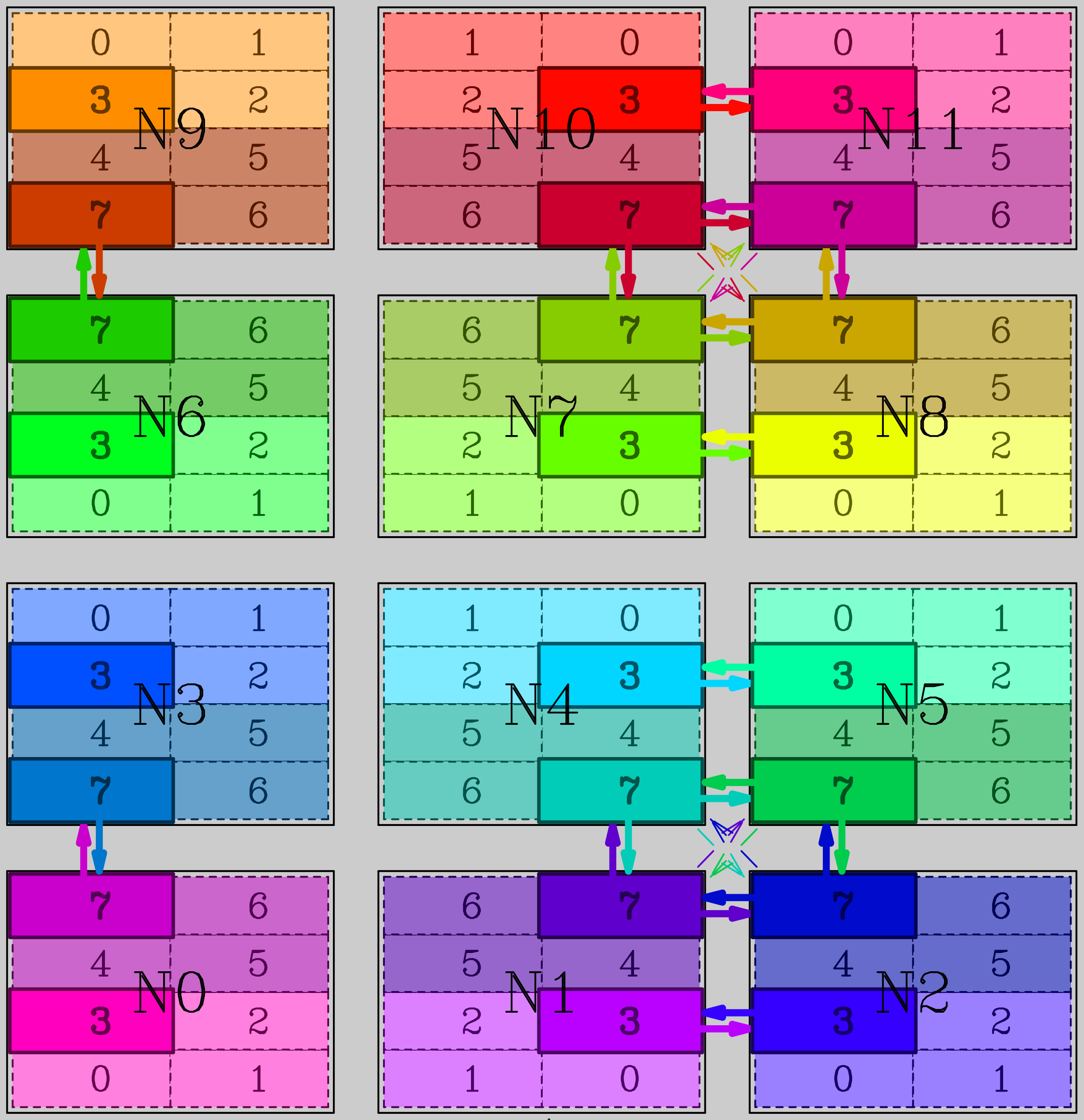
12 MPI nodes, 8 tiles, 2 threads, continued...



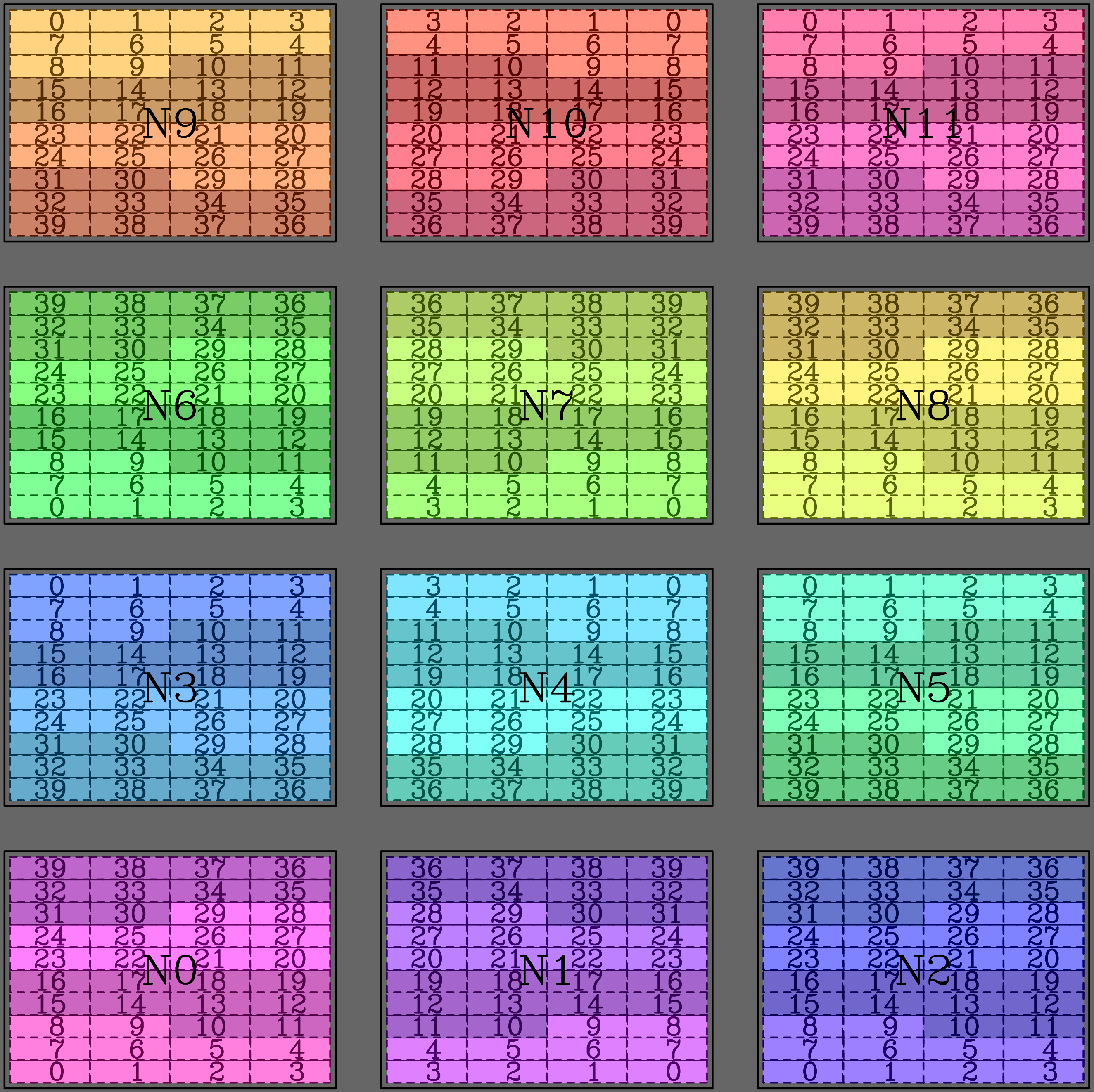
12 MPI nodes, 8 tiles, 2 threads, continued...

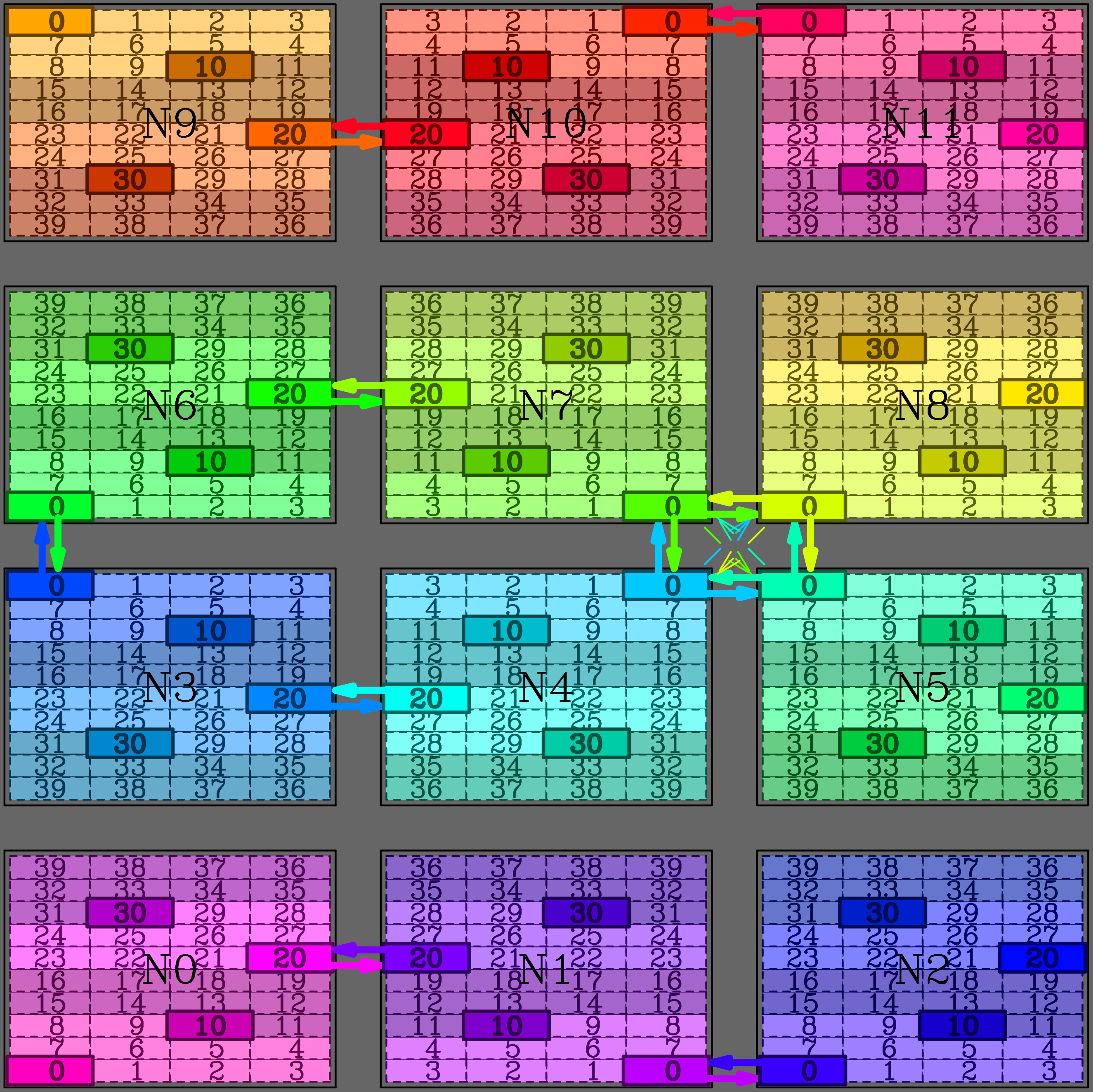


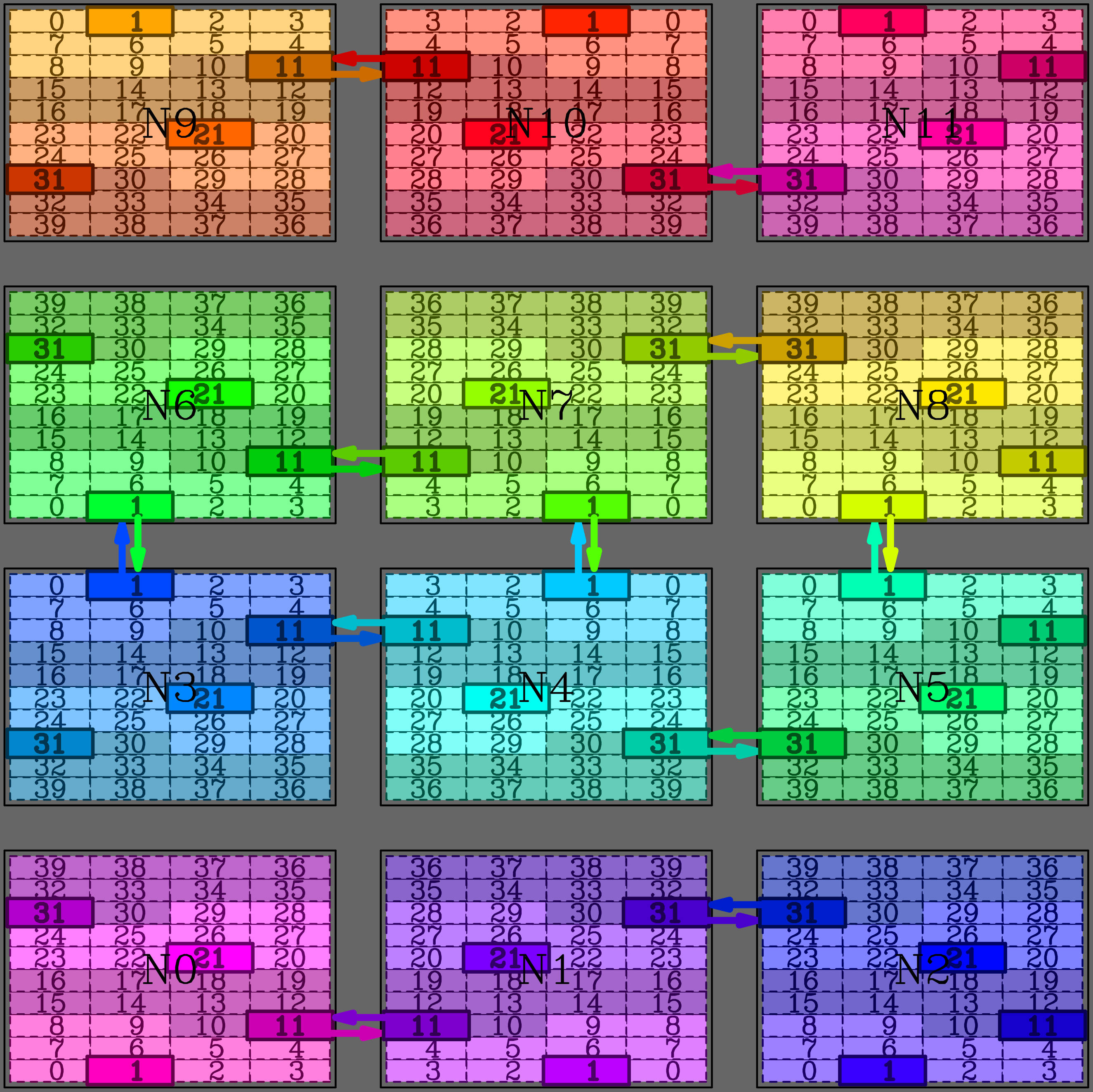
12 MPI nodes, 8 tiles, 2 threads, continued...

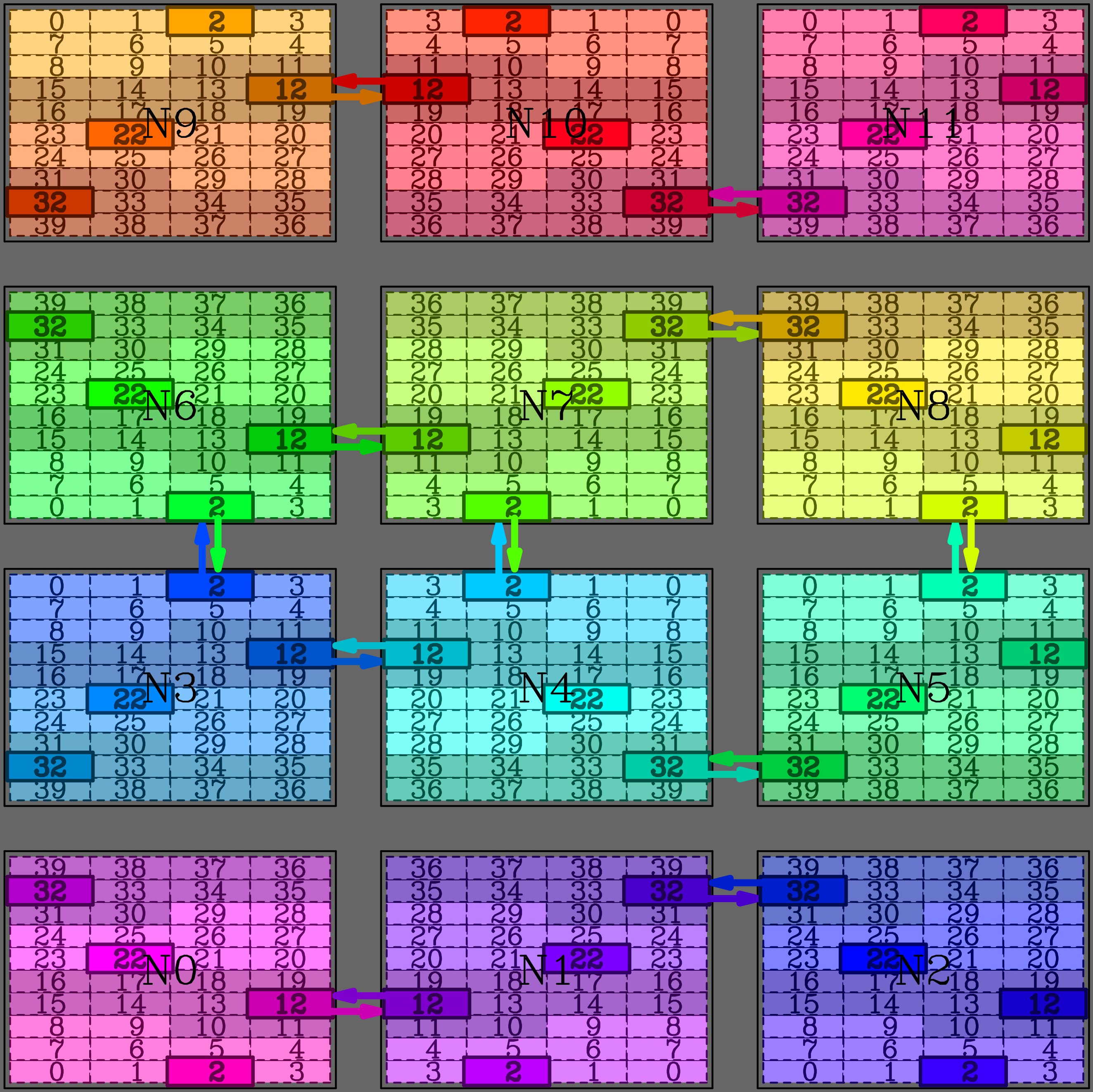


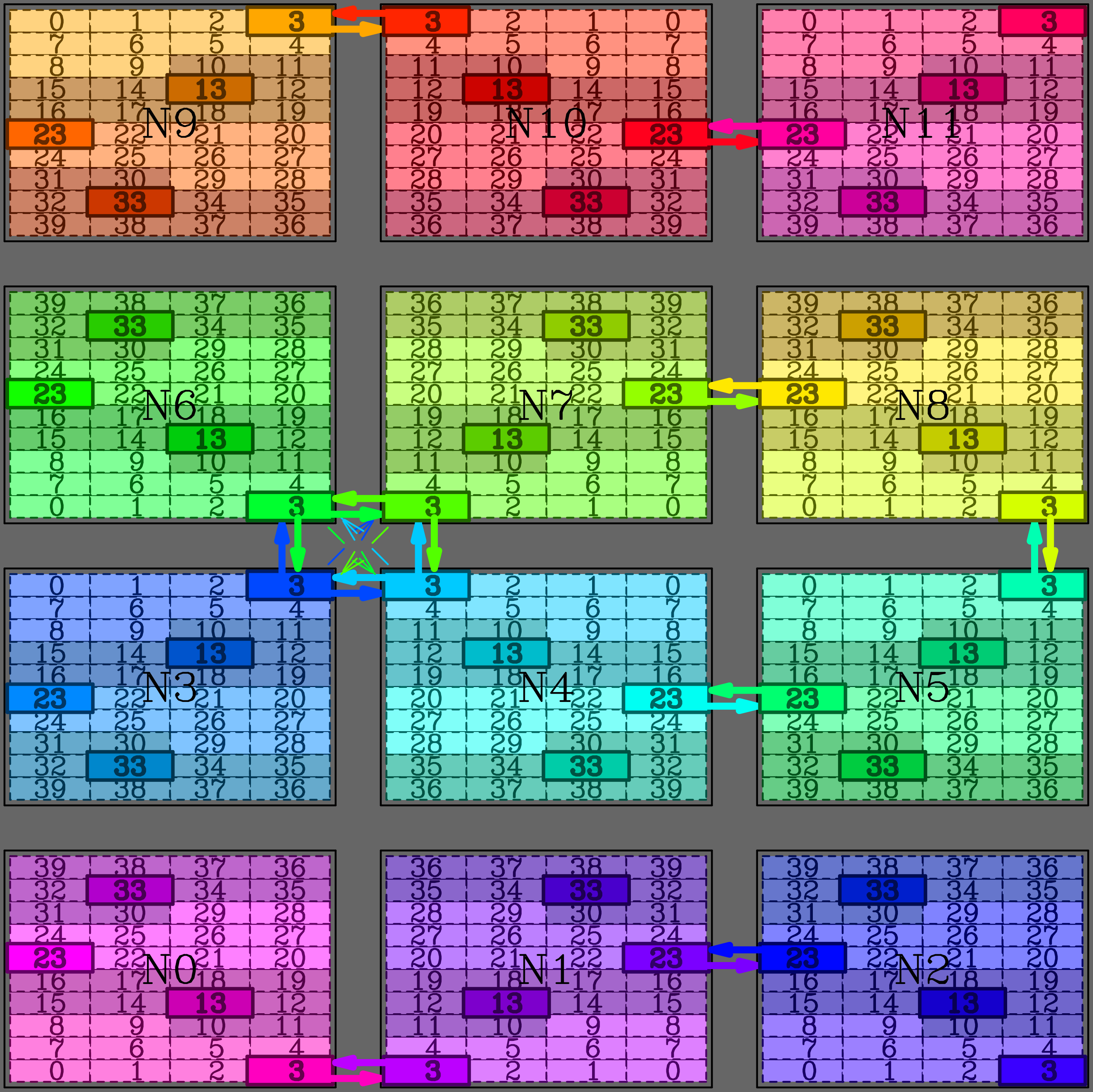
Scheduling messages by threads: A more compicated case

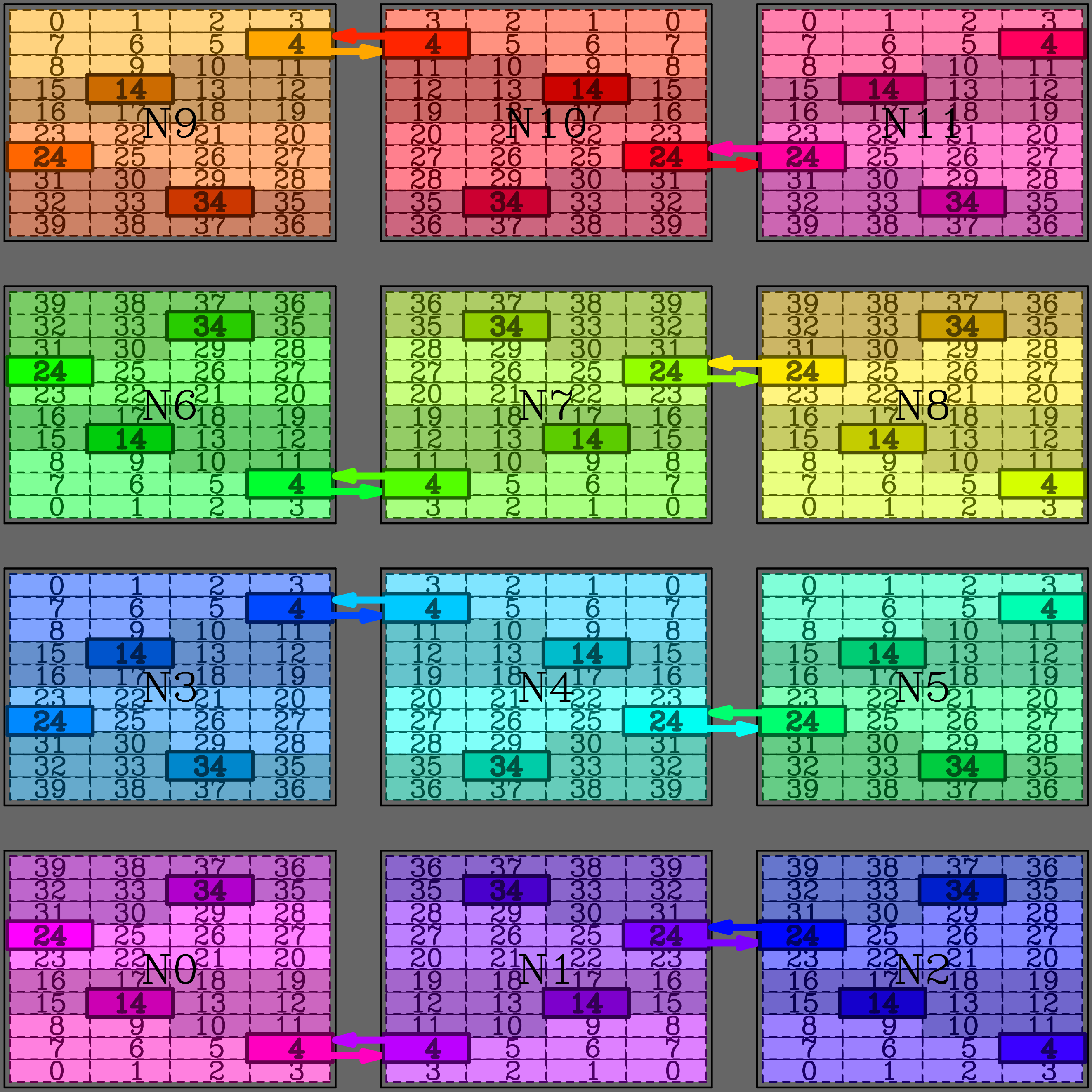

N9
N10
N11
N6
N7
N8
N3
N4
N5
N0
N1
N2

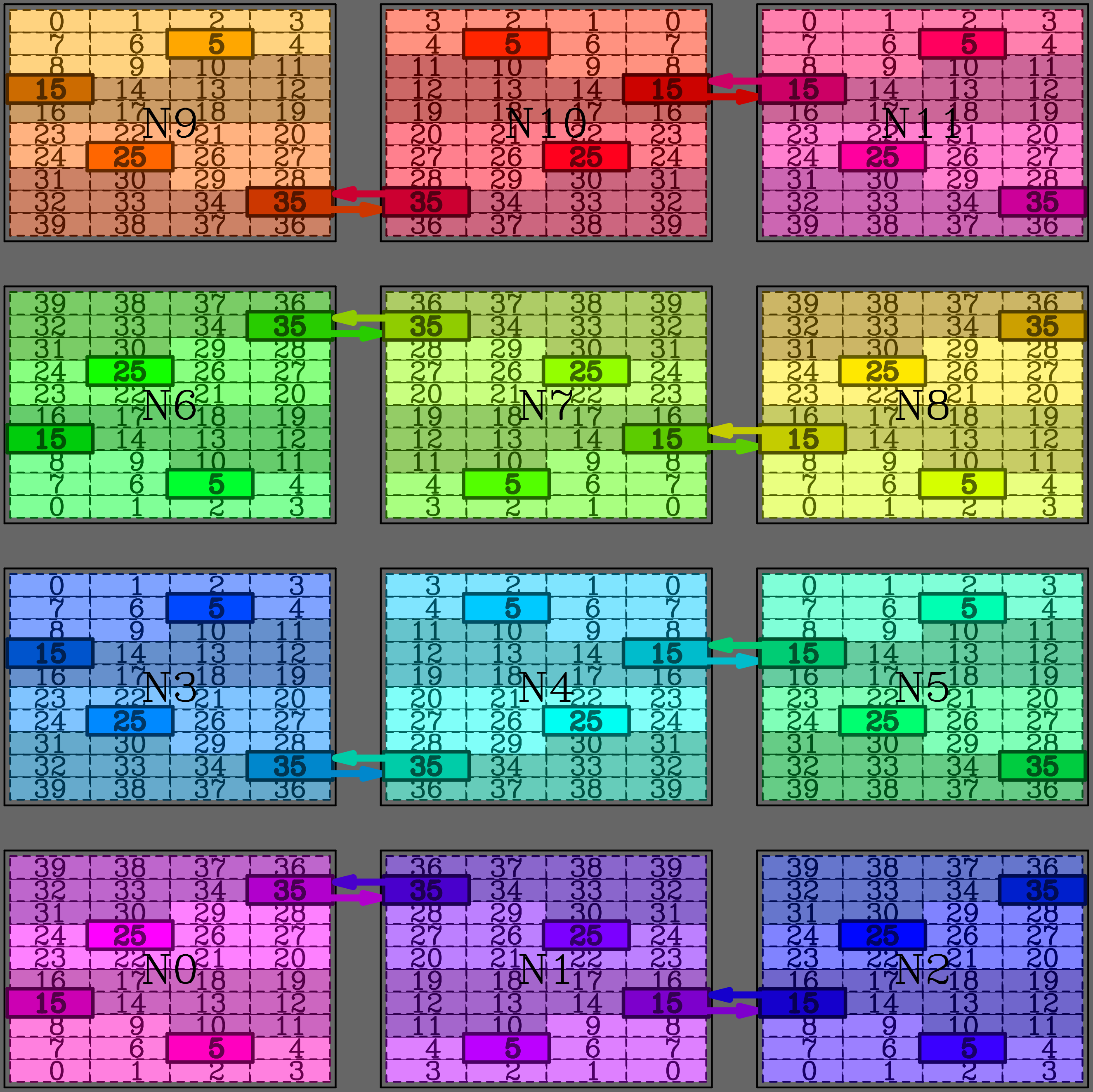

N9
N10
N11
N6
N7
N8
N3
N4
N5
N0
N1
N2

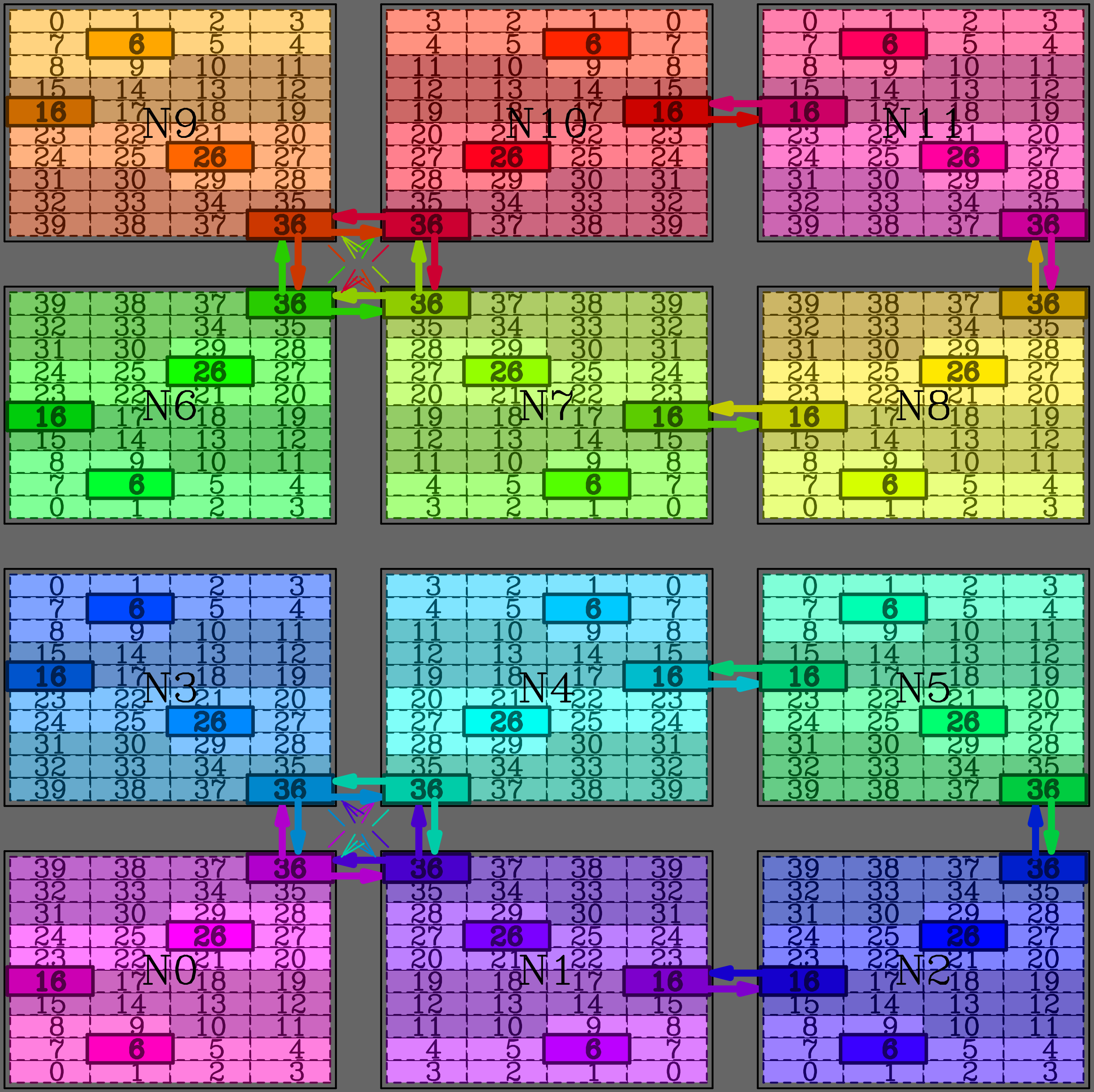

N9
N10
N11
N6
N7
N8
N3
N4
N5
N0
N1
N2

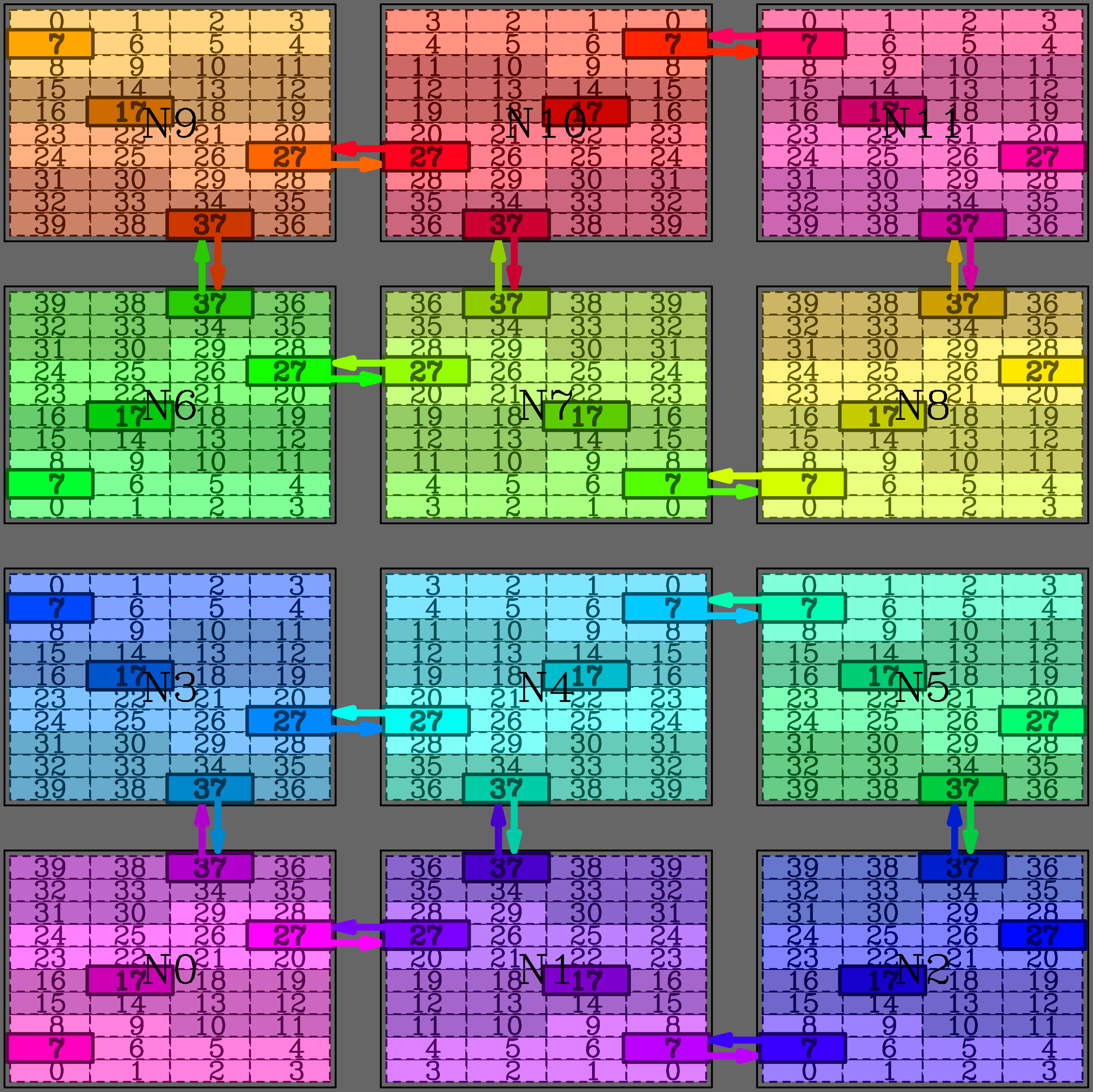

N9
N10
N11
N6
N7
N8
N3
N4
N5
N0
N1
N2

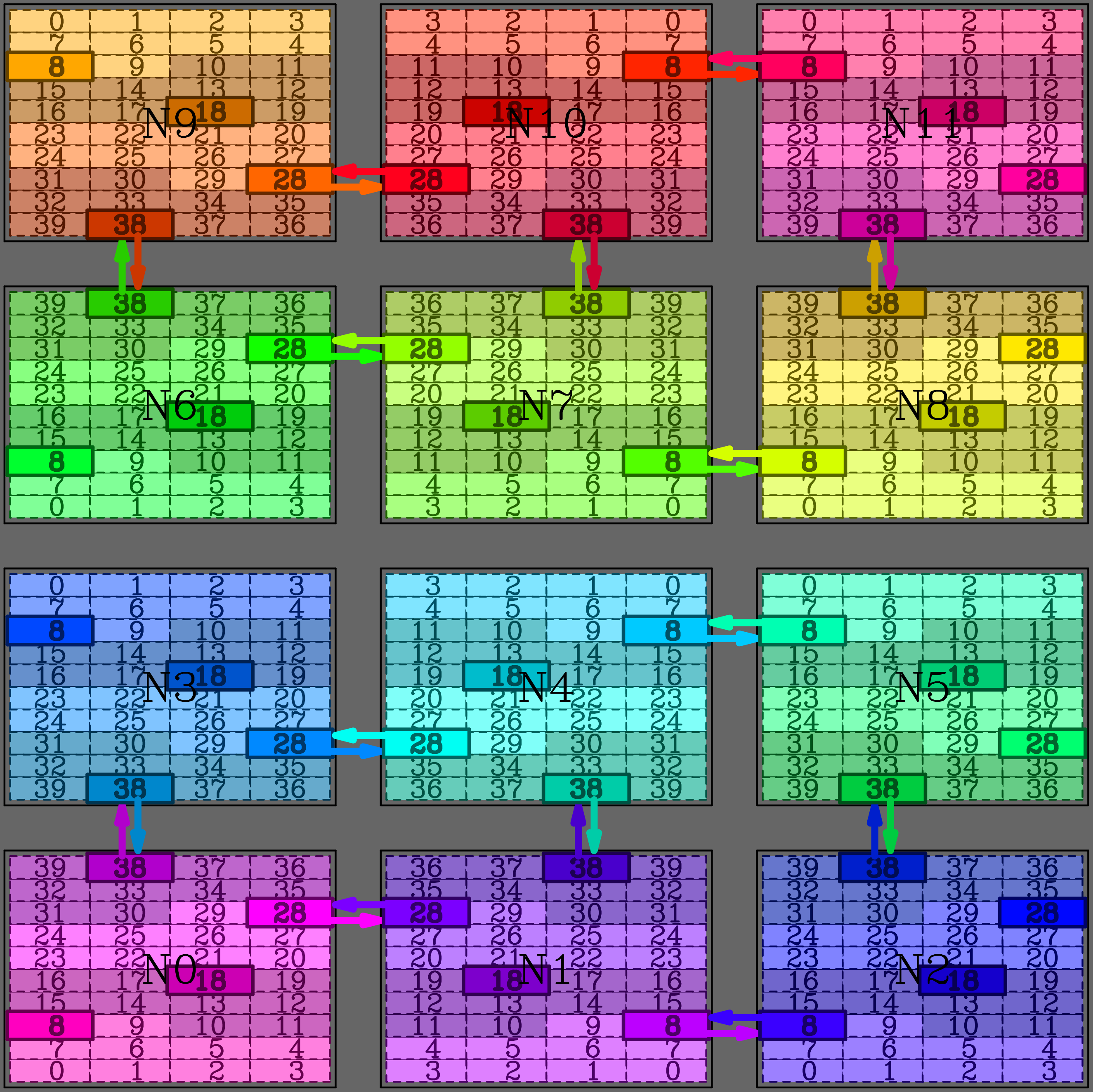

N9
N10
N11
N6
N7
N8
N3
N4
N5
N0
N1
N2

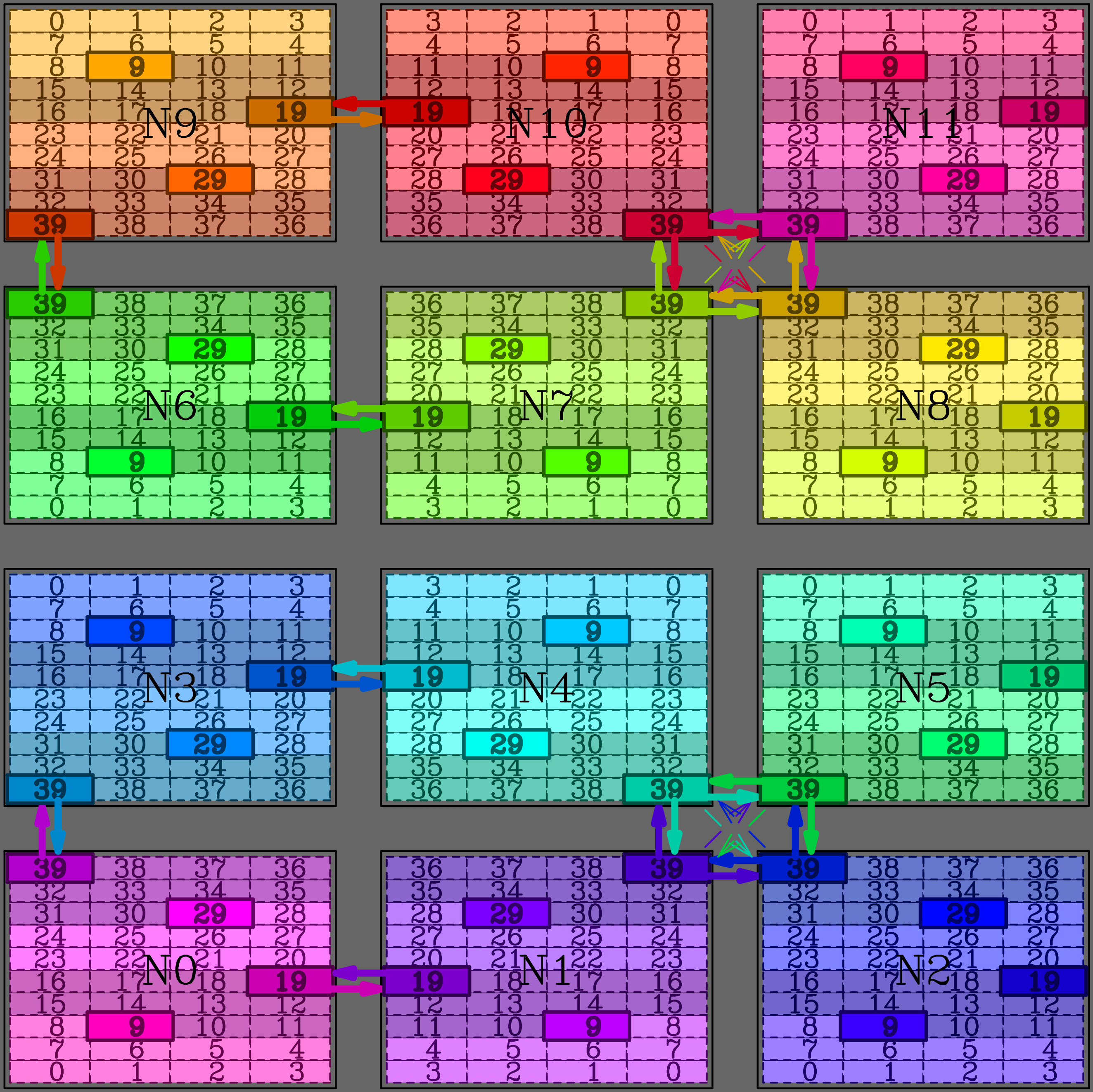

N0
N1
N2
N3
N4
N5
N6
N7
N8
N9
N10
N11


N9
N10
N11
N6
N7
N8
N3
N4
N5
N0
N1
N2


N9
N10
N11
N6
N7
N8
N3
N4
N5
N0
N1
N2


N9
N10
N11
N6
N7
N8
N3
N4
N5
N0
N1
N2


N9
N10
N11
N6
N7
N8
N3
N4
N5
N0
N1
N2


N9
N10
N11
N6
N7
N8
N3
N4
N5
N0
N1
N2

Conclusion

- Besides cache blocking, inner tiling can be used to control scheduling of communications on order to alleviate competition between/among CPUs for network access
- works in ”soft mode”: threads are free-running, and there are no barriers around MPI calls, and no extra barriers relatively to the original OpenMP-only code.
- **thread trajectory does matter** by itself, and relatively to other threads, and it is no longer a simple zig-zag pattern
- Overlaps computations and communications in time
- Soften peak loads on network switch
- Finer tiling also means smaller MPI-messages, which negatively affects performance. Compromise is needed, and preliminary experience tells that message scheduling is more critical than cache optimization

Does it work?

- Yes, in sense that the code is functional and produces correct result
- Yes, we can exceed performance of pure MPI code on dual-CPU nodes connected. via GigE network for problems of our interest
- Too early to say about quad-Core nodes: in comparison with our dual-Opteron cluster nodes, the quad-Core provides 3 times more CPU power, but only 5/6 of aggregate memory bandwidth, and only a single NIC card, ultimately making it a harder problem.

Optimally tiled OpenMP code scales as $\times 1.9$ then $\times 1.4...1.5$ for a large out-of cache problem when going $1 \to 2$ then $2 \to 4$ cores, well off-setting an extra cost of quad-relatively to dual-core.

- Performance of dream code is still out of reach: GigE may be simply too slow, or we have to go for a more ambitious problem
- Looking back: never trust you intuition about ideas for improving performance of MPI code: always try them
- Swappable MPI halo-exchange routines