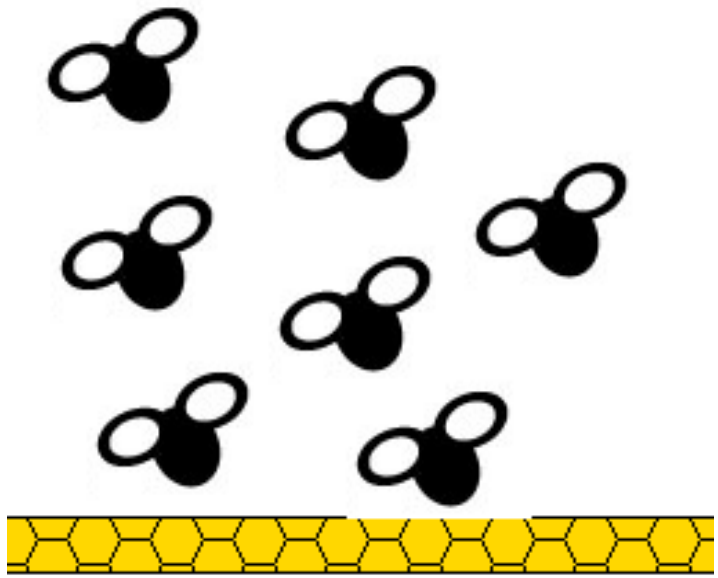
Parallel Algorithms for the Solution and Estimation of DSGE Models

Markus Krätzig[1] Viktor Winschel[2]

[1]Humboldt-Universität zu Berlin, Germany
[2]University of Mannheim, Germany

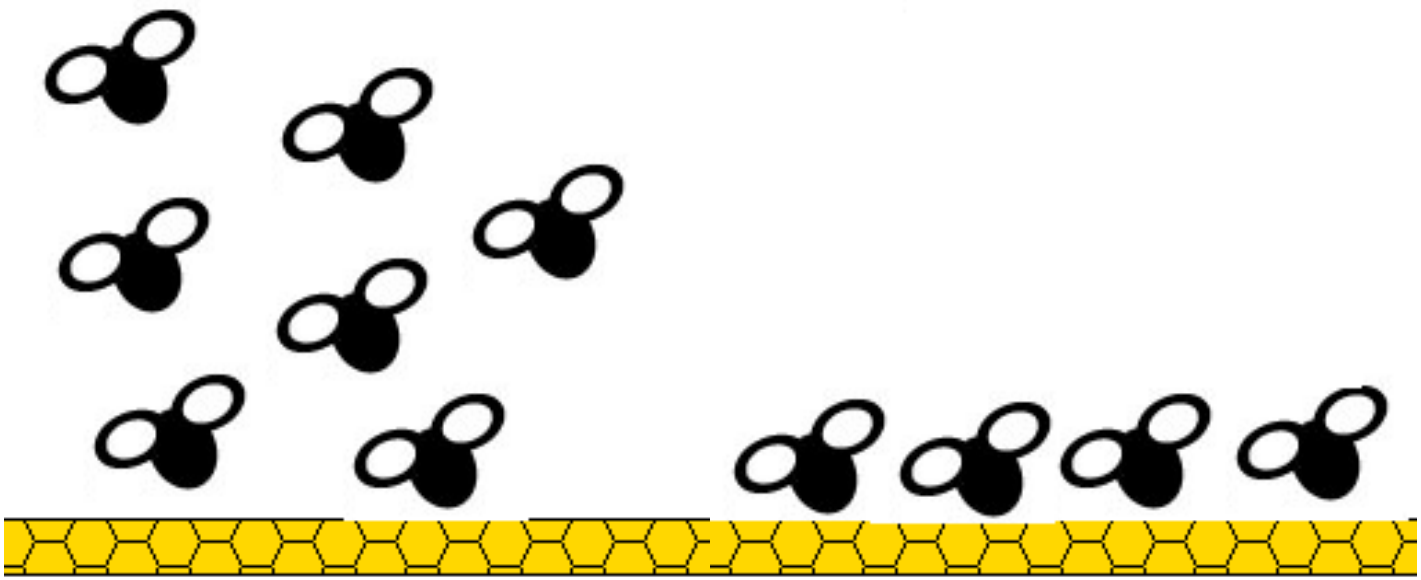
June 27, 2008

Threads Ideal



A thread is a call sequence the executes independently of other threads. It may share underlying system resources with other threads.

Threads Synchronized



Model Class

$f(s, x, \mathrm{E}h(s, x, e', s', x')) = 0$ $s' = g(s, x, e')$	
Specified Functions	
f	equilibrium conditions
h	expectation functions
g	state transitions
Variables	
s	state variables
x	policy variables
z	expectation variables
e	stochastic shocks

Standard RBC Model for N Countries

$$\max U\{\{c_{n,t}, l_{n,t}, i_{n,t}\}_{n=1}^{N}\}_{t=0}^{\infty}$$

$$\begin{aligned}
U &= E_0 \sum_{t=0}^{\infty} \sum_{n=1}^{N} \beta^t U_{n,t} \\
U_{n,t} &= \frac{(c_{n,t}^{\theta_n}(1-l_{n,t})^{1-\theta_n})^{1-\tau_n}}{1-\tau_n} \\
\sum_{n=1}^{N}(y_{n,t} - c_{n,t} - i_{n,t}) &= 0 \\
y_{n,t} &= e^{a_{n,t}} k_{n,t}^{\alpha_n} l_{n,t}^{1-\alpha_n} \\
k_{n,t+1} &= (1-\delta_n)k_{n,t} + i_{n,t} - 0.5\kappa_n i_{n,t}^2 \\
a_{n,t+1} &= \rho_n a_{n,t} + e_{n,t+1}
\end{aligned}$$

Solution

Solutions can be characterized as policy functions

$$x_t = x^*(s_t)$$

with $x_t = (c_t, l_t)'$ and $s_t = (k_t, a_t)'$

In general no analytic solution is possible. Global polynomial approximation (Judd, 1992) may be used (or linearization, pertubation...).

Approximation Strategy

Policy functions by Chebychev polynomials:

$$x = \Psi(s) = \sum_{i=1}^{n} c_i \psi_i(s)$$

Expectations over e' via Gauss Quadrature:

$$\begin{aligned} \mathrm{E}h(s, x, e', s', x') &= \\ &= \sum_j \omega_j h(s, x^*(s), e'_j, s'_j, x^*(s'_j)) \end{aligned}$$

over optimal nodes and weights. We use the Smolyak operator (Smolyak, 1963) to cope with the curse of dimensionality.

Solving by Function Iteration

0. find the initial policy: $x^{(0)}$ at states grid s with G grid points

Solving by Function Iteration

0. find the initial policy: $x^{(0)}$ at states grid s with G grid points
1. approximate the policy function at states grid and find Chebychev interpolant $c^{(k)} = B(s)^{-1}x^{(k)}$

Solving by Function Iteration

0. find the initial policy: $x^{(0)}$ at states grid s with G grid points
1. approximate the policy function at states grid and find Chebychev interpolant $c^{(k)} = B(s)^{-1}x^{(k)}$
2. rational expectations:

Solving by Function Iteration

0. find the initial policy: $x^{(0)}$ at states grid s with G grid points
1. approximate the policy function at states grid and find Chebychev interpolant $c^{(k)} = B(s)^{-1}x^{(k)}$
2. rational expectations:
   - 2.1 **for all discrete shock realizations from the quadrature rule** $j = 1, ..., J$

Solving by Function Iteration

0. find the initial policy: $x^{(0)}$ at states grid s with G grid points
1. approximate the policy function at states grid and find Chebychev interpolant $c^{(k)} = B(s)^{-1}x^{(k)}$
2. rational expectations:
   - 2.1 **for all discrete shock realizations from the quadrature rule** $j = 1, ..., J$
     - 2.1.1 **calculate the state transitions:** $s'_j = g(s, x^{(k)}, e'_j)$

Solving by Function Iteration

0. find the initial policy: $x^{(0)}$ at states grid s with G grid points
1. approximate the policy function at states grid and find Chebychev interpolant $c^{(k)} = B(s)^{-1}x^{(k)}$
2. rational expectations:
   - 2.1 **for all discrete shock realizations from the quadrature rule** $j = 1, ..., J$
     - 2.1.1 **calculate the state transitions:** $s'_j = g(s, x^{(k)}, e'_j)$
     - 2.1.2 **interpolate the next policy using the Chebychev interpolant:** $x'_j = B(s'_j)c^{(k)}$

Solving by Function Iteration

0. find the initial policy: $x^{(0)}$ at states grid s with G grid points
1. approximate the policy function at states grid and find Chebychev interpolant $c^{(k)} = B(s)^{-1}x^{(k)}$
2. rational expectations:
   - 2.1 **for all discrete shock realizations from the quadrature rule** $j = 1, ..., J$
     - 2.1.1 **calculate the state transitions:** $s'_j = g(s, x^{(k)}, e'_j)$
     - 2.1.2 **interpolate the next policy using the Chebychev interpolant:** $x'_j = B(s'_j)c^{(k)}$
   - 2.2 evaluate the expectations with Gaussian quadrature: $z = \sum_j w_j h(s, x, e'_j, s'_j, x'_j)$.

Solving by Function Iteration

0. find the initial policy: $x^{(0)}$ at states grid s with G grid points
1. approximate the policy function at states grid and find Chebychev interpolant $c^{(k)} = B(s)^{-1}x^{(k)}$
2. rational expectations:
   - 2.1 **for all discrete shock realizations from the quadrature rule** $j = 1, ..., J$
     - 2.1.1 **calculate the state transitions:** $s'_j = g(s, x^{(k)}, e'_j)$
     - 2.1.2 **interpolate the next policy using the Chebychev interpolant:** $x'_j = B(s'_j)c^{(k)}$
   - 2.2 evaluate the expectations with Gaussian quadrature: $z = \sum_j w_j h(s, x, e'_j, s'_j, x'_j)$.
3. function iteration:

Solving by Function Iteration

0. find the initial policy: $x^{(0)}$ at states grid s with G grid points
1. approximate the policy function at states grid and find Chebychev interpolant $c^{(k)} = B(s)^{-1}x^{(k)}$
2. rational expectations:
   - 2.1 **for all discrete shock realizations from the quadrature rule** $j = 1, ..., J$
     - 2.1.1 **calculate the state transitions:** $s'_j = g(s, x^{(k)}, e'_j)$
     - 2.1.2 **interpolate the next policy using the Chebychev interpolant:** $x'_j = B(s'_j)c^{(k)}$
   - 2.2 evaluate the expectations with Gaussian quadrature: $z = \sum_j w_j h(s, x, e'_j, s'_j, x'_j)$.
3. function iteration:
   - 3.1 **for all grid points** i**: solve for** x_i^* **in** $f(s_i, x_i^*, z_i(x^{(k)})) = 0$ **given** $x^{(k)}$

Solving by Function Iteration

0. find the initial policy: $x^{(0)}$ at states grid s with G grid points
1. approximate the policy function at states grid and find Chebychev interpolant $c^{(k)} = B(s)^{-1}x^{(k)}$
2. rational expectations:
   - 2.1 **for all discrete shock realizations from the quadrature rule** $j = 1, ..., J$
     - 2.1.1 **calculate the state transitions:** $s'_j = g(s, x^{(k)}, e'_j)$
     - 2.1.2 **interpolate the next policy using the Chebychev interpolant:** $x'_j = B(s'_j)c^{(k)}$
   - 2.2 evaluate the expectations with Gaussian quadrature: $z = \sum_j w_j h(s, x, e'_j, s'_j, x'_j)$.
3. function iteration:
   - 3.1 **for all grid points** i**: solve for** x_i^* **in** $f(s_i, x_i^*, z_i(x^{(k)})) = 0$ **given** $x^{(k)}$
   - 3.2 update: $x^{(k+1)} = x^* = (x_1^*, \ldots, x_G^*)'$

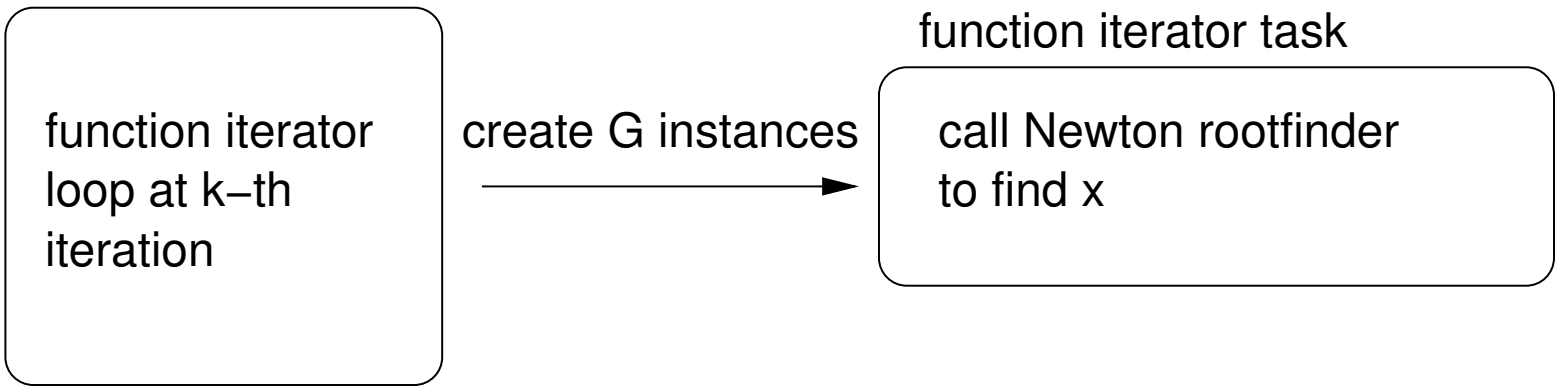
Solving by Function Iteration

0. find the initial policy: $x^{(0)}$ at states grid s with G grid points
1. approximate the policy function at states grid and find Chebychev interpolant $c^{(k)} = B(s)^{-1}x^{(k)}$
2. rational expectations:
   - 2.1 **for all discrete shock realizations from the quadrature rule** $j = 1, ..., J$
     - 2.1.1 **calculate the state transitions:** $s'_j = g(s, x^{(k)}, e'_j)$
     - 2.1.2 **interpolate the next policy using the Chebychev interpolant:** $x'_j = B(s'_j)c^{(k)}$
   - 2.2 evaluate the expectations with Gaussian quadrature: $z = \sum_j w_j h(s, x, e'_j, s'_j, x'_j)$.
3. function iteration:
   - 3.1 **for all grid points** i**: solve for** x_i^* **in** $f(s_i, x_i^*, z_i(x^{(k)})) = 0$ **given** $x^{(k)}$
   - 3.2 update: $x^{(k+1)} = x^* = (x_1^*, \ldots, x_G^*)'$
   - 3.3 calculate the residual: $R = ||x^{k+1} - x^{(k)}||_\infty$

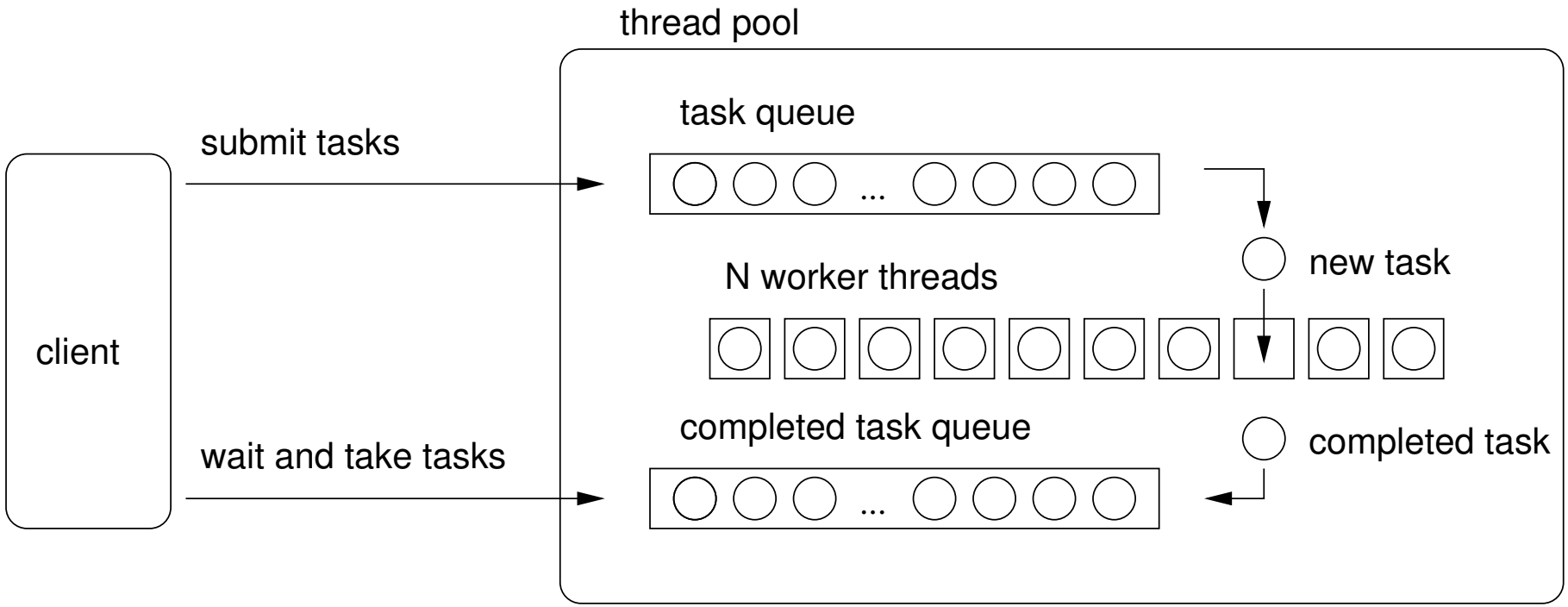
Solving by Function Iteration

0. find the initial policy: $x^{(0)}$ at states grid s with G grid points
1. approximate the policy function at states grid and find Chebychev interpolant $c^{(k)} = B(s)^{-1}x^{(k)}$
2. rational expectations:
   - 2.1 **for all discrete shock realizations from the quadrature rule** $j = 1, ..., J$
     - 2.1.1 **calculate the state transitions:** $s'_j = g(s, x^{(k)}, e'_j)$
     - 2.1.2 **interpolate the next policy using the Chebychev interpolant:** $x'_j = B(s'_j)c^{(k)}$
   - 2.2 evaluate the expectations with Gaussian quadrature: $z = \sum_j w_j h(s, x, e'_j, s'_j, x'_j)$.
3. function iteration:
   - 3.1 **for all grid points** i**: solve for** x^*_i **in** $f(s_i, x^*_i, z_i(x^{(k)})) = 0$ **given** $x^{(k)}$
   - 3.2 update: $x^{(k+1)} = x^* = (x^*_1, \ldots, x^*_G)'$
   - 3.3 calculate the residual: $R = ||x^{k+1} - x^{(k)}||_\infty$
4. k=k+1, go to (1) until $R \approx 0$

Parallelization



Thread Pool Pattern



implemented in *java.util.concurrent* package and many others

State Space

The model solving policy function $x(s)$ can be used to substitute out the policy variables in the state transition equation.

Augmented with a measurement equation we obtain the model's empirical implication for the observables in a nonlinear state space model

$$\begin{aligned} s_t &= g(s_{t-1}, x(s_{t-1}), e_t) = g^*(s_{t-1}, e_t) \\ y_t &= m(s_t) + \epsilon_t. \end{aligned}$$

With distributional assumptions for the state and measurement shocks, e_t and ϵ_t, the equations can be cast in terms of state and measurement densities

$$p(s_t|s_{t-1})$$

$$p(y_t|s_t),$$

respectively.

The filtering approach evaluates the likelihood of this model by the prediction and filtering step:

$$\mathcal{L}(\theta; y_{1:T}) = p(y_{1:T}|\theta) = \prod_{t=1}^{T} p(y_t|y_{1:t-1}, \theta) = \prod_{t=1}^{T} l_t.$$

Parallel Metropolis-Hastings Algorithm with Mixing

1. Choose start values $\hat{\theta}_{1,m}$ for all $m = 1, ..., M$ and b, γ^{GE} for an acceptance ratio of $\approx 30\%$

Parallel Metropolis-Hastings Algorithm with Mixing

1. Choose start values $\hat{\theta}_{1,m}$ for all $m = 1, ..., M$ and b, γ^{GE} for an acceptance ratio of $\approx 30\%$
2. For $n = 1$, while $n - J < N$, n=n+1

Parallel Metropolis-Hastings Algorithm with Mixing

1. Choose start values $\hat{\theta}_{1,m}$ for all $m = 1, ..., M$ and b, γ^{GE} for an acceptance ratio of $\approx 30\%$
2. For $n = 1$, while $n - J < N$, n=n+1
   2.1 Repeat for $m = 1, ..., M$

Parallel Metropolis-Hastings Algorithm with Mixing

1. Choose start values $\hat{\theta}_{1,m}$ for all $m = 1, ..., M$ and b, γ^{GE} for an acceptance ratio of $\approx 30\%$
2. For $n = 1$, while $n - J < N$, n=n+1
   - 2.1 Repeat for $m = 1, ..., M$
     - 2.1.1 Draw m_1 and m_2 such that $m_1 \neq m_2 \neq m$

Parallel Metropolis-Hastings Algorithm with Mixing

1. Choose start values $\hat{\theta}_{1,m}$ for all $m = 1, ..., M$ and b, γ^{GE} for an acceptance ratio of $\approx 30\%$
2. For $n = 1$, while $n - J < N$, n=n+1
   - 2.1 Repeat for $m = 1, ..., M$
     - 2.1.1 Draw m_1 and m_2 such that $m_1 \neq m_2 \neq m$
     - 2.1.2 Candidate: $\hat{\theta}^*_m = \hat{\theta}_{m,n} + \gamma^{GE}(\hat{\theta}_{m_1,n} - \hat{\theta}_{m_2,n}) + \epsilon$, $\mathcal{N}(\epsilon; 0, bI)$

Parallel Metropolis-Hastings Algorithm with Mixing

1. Choose start values $\hat{\theta}_{1,m}$ for all $m = 1, ..., M$ and b, γ^{GE} for an acceptance ratio of $\approx 30\%$
2. For $n = 1$, while $n - J < N$, n=n+1
   - 2.1 Repeat for $m = 1, ..., M$
     - 2.1.1 Draw m_1 and m_2 such that $m_1 \neq m_2 \neq m$
     - 2.1.2 Candidate: $\hat{\theta}^*_m = \hat{\theta}_{m,n} + \gamma^{GE}(\hat{\theta}_{m_1,n} - \hat{\theta}_{m_2,n}) + \epsilon$, $\mathcal{N}(\epsilon; 0, bI)$
   - 2.2 **Repeat for** $m = 1, ..., M$

Parallel Metropolis-Hastings Algorithm with Mixing

1. Choose start values $\hat{\theta}_{1,m}$ for all $m = 1, ..., M$ and b, γ^{GE} for an acceptance ratio of $\approx 30\%$
2. For $n = 1$, while $n - J < N$, n=n+1
   - 2.1 Repeat for $m = 1, ..., M$
     - 2.1.1 Draw m_1 and m_2 such that $m_1 \neq m_2 \neq m$
     - 2.1.2 Candidate: $\hat{\theta}^*_m = \hat{\theta}_{m,n} + \gamma^{GE}(\hat{\theta}_{m_1,n} - \hat{\theta}_{m_2,n}) + \epsilon$, $\mathcal{N}(\epsilon; 0, bI)$
   - 2.2 **Repeat for** $m = 1, ..., M$
     - 2.2.1 **Acceptance:**

$$\hat{\theta}_{n,m} = \begin{cases} \hat{\theta}^*_m & \textbf{if } U(0,1) \leq \frac{p(y_{1:t}|\hat{\theta}^*_m)p(\hat{\theta}^*_m)}{p(y_{1:t}|\hat{\theta}_m)p(\hat{\theta}_m)} \\ \hat{\theta}_{n,m} & \textbf{otherwise} \end{cases}$$

Parallel Metropolis-Hastings Algorithm with Mixing

1. Choose start values $\hat{\theta}_{1,m}$ for all $m = 1, ..., M$ and b, γ^{GE} for an acceptance ratio of $\approx 30\%$
2. For $n = 1$, while $n - J < N$, n=n+1
   - 2.1 Repeat for $m = 1, ..., M$
     - 2.1.1 Draw m_1 and m_2 such that $m_1 \neq m_2 \neq m$
     - 2.1.2 Candidate: $\hat{\theta}^*_m = \hat{\theta}_{m,n} + \gamma^{GE}(\hat{\theta}_{m_1,n} - \hat{\theta}_{m_2,n}) + \epsilon$, $\mathcal{N}(\epsilon; 0, bI)$
   - 2.2 **Repeat for** $m = 1, ..., M$
     - 2.2.1 **Acceptance:**

$$\hat{\theta}_{n,m} = \begin{cases} \hat{\theta}^*_m & \textbf{if } U(0,1) \leq \frac{p(y_{1:t}|\hat{\theta}^*_m)p(\hat{\theta}^*_m)}{p(y_{1:t}|\hat{\theta}_m)p(\hat{\theta}_m)} \\ \hat{\theta}_{n,m} & \textbf{otherwise} \end{cases}$$

   - 2.3 Decide on J by diagnostic tests

Parallel Metropolis-Hastings Algorithm with Mixing

1. Choose start values $\hat{\theta}_{1,m}$ for all $m = 1, ..., M$ and b, γ^{GE} for an acceptance ratio of $\approx 30\%$
2. For $n = 1$, while $n - J < N$, n=n+1
   - 2.1 Repeat for $m = 1, ..., M$
     - 2.1.1 Draw m_1 and m_2 such that $m_1 \neq m_2 \neq m$
     - 2.1.2 Candidate: $\hat{\theta}^*_m = \hat{\theta}_{m,n} + \gamma^{GE}(\hat{\theta}_{m_1,n} - \hat{\theta}_{m_2,n}) + \epsilon$, $\mathcal{N}(\epsilon; 0, bI)$
   - 2.2 **Repeat for** $m = 1, ..., M$
     - 2.2.1 **Acceptance:**

$$\hat{\theta}_{n,m} = \begin{cases} \hat{\theta}^*_m & \textbf{if } U(0,1) \leq \frac{p(y_{1:t}|\hat{\theta}^*_m)p(\hat{\theta}^*_m)}{p(y_{1:t}|\hat{\theta}_m)p(\hat{\theta}_m)} \\ \hat{\theta}_{n,m} & \textbf{otherwise} \end{cases}$$

   - 2.3 Decide on J by diagnostic tests
3. Disregard burn-in draws $\hat{\theta}_{1:J,1:M}$

Simulation Setup for Nonlinear Solution

Cluster with 16 Xeon CPUs at 2.7GHz (but only 14 can are free);
GNU/Linux openSUSE 10.2;
Java virtual machine 1.6.0_06

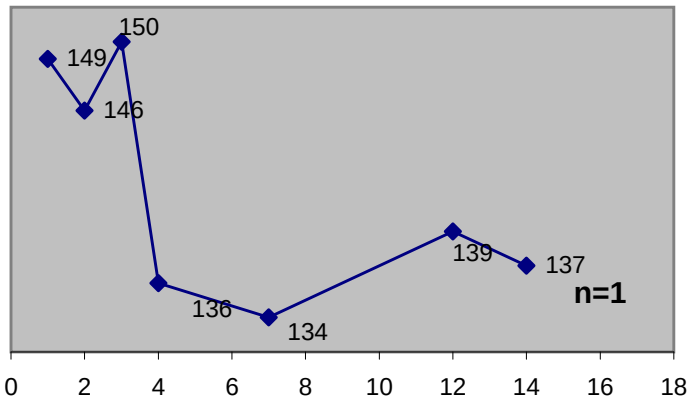
- parameters:
  $\alpha_i = 0.4, \beta = 0.99, \delta_i = 0.02, \kappa_i = 0.01, \rho_i = 0.95, \tau_i = 2$
  $\theta_i = 0.357, \sigma_{a_i} = 0.007$
- states grid bounds:
  $a_i \in [-0.06; 0.06], \quad k_i \in [15; 30]$
- approximation level $= 3$, integration level $= 3$
- tolerance for function iterator: 1×10^{-4}
- starting values for steady state:
  $a_i = 0, i_i = 0.5, k_i = 23, l_i = 0.3$
- strong convergence tolerance for Newton rootfinder: 1×10^{-10}
- set thread pool size to 1, 2, 3, 4, 7, 12, 14

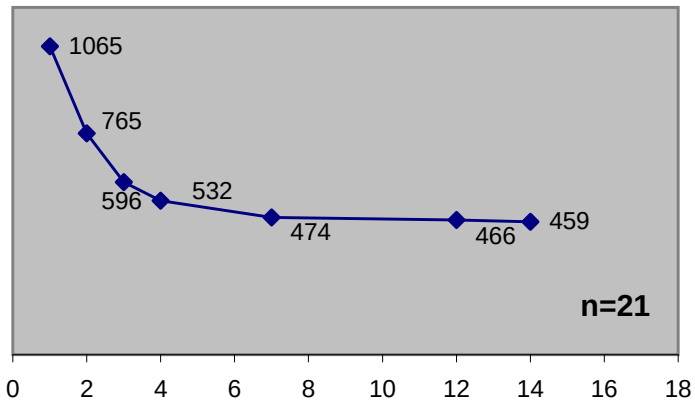
Amdahl's Law

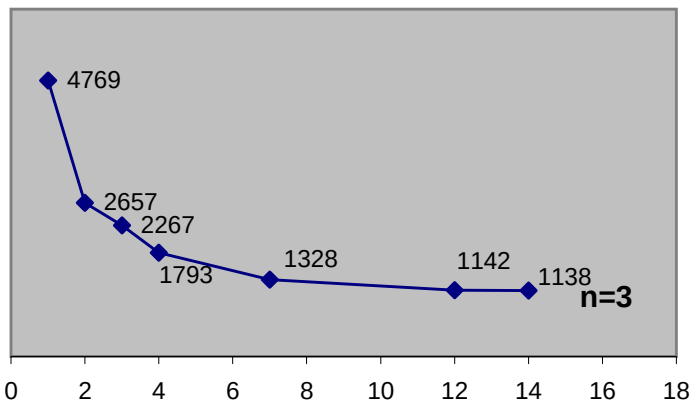
$$\sup speedup = \frac{1}{F + (1 - F)/N}$$

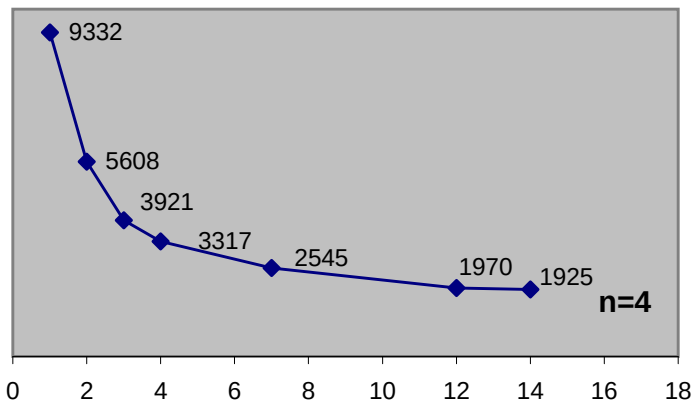
- F is percentage of system that cannot be parallelized
- N is number of processors
- provides a theoretical upper bound for the expected speedup of using N processors instead of one

Nonlin. Sol.: Computing Times vs. # Processors

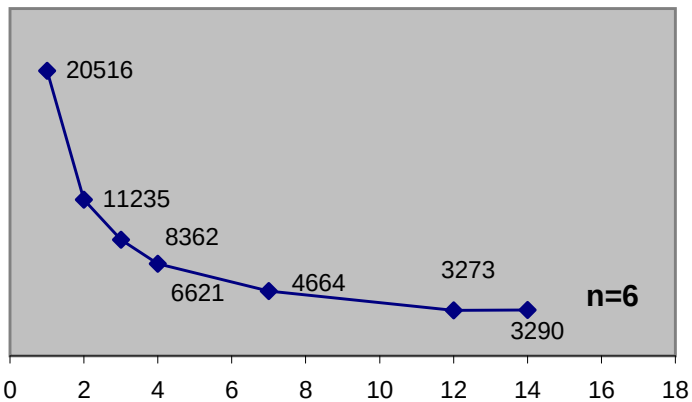


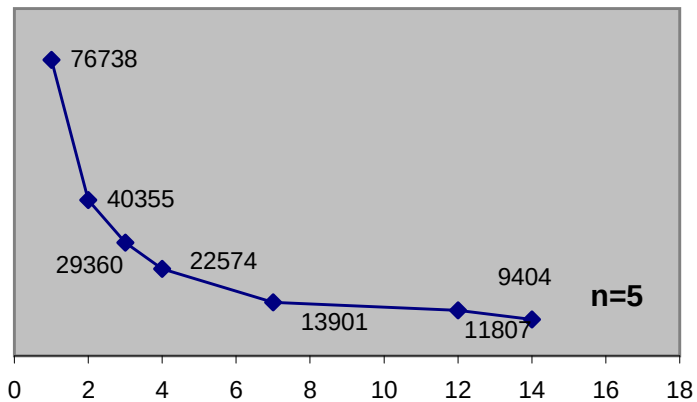


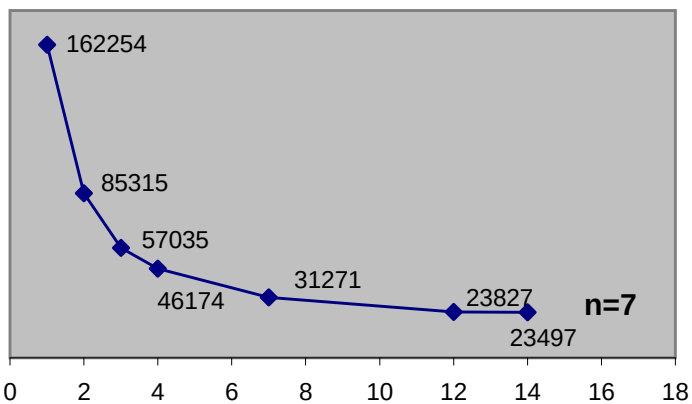




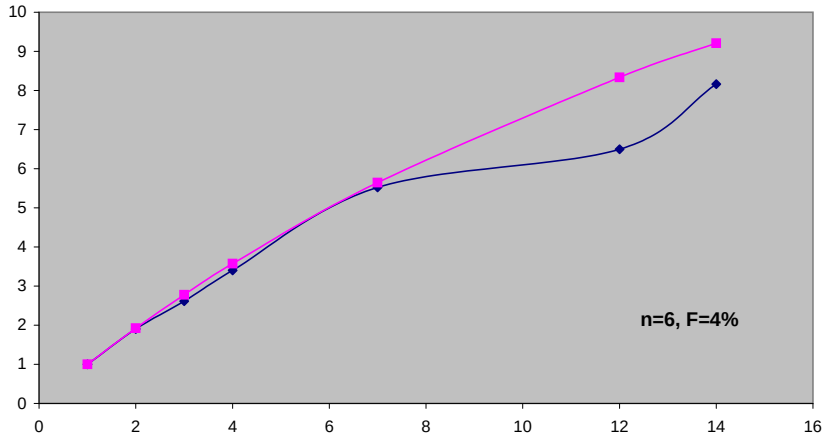
Nonlin. Sol.: Computing Times vs. # Processors cont'd

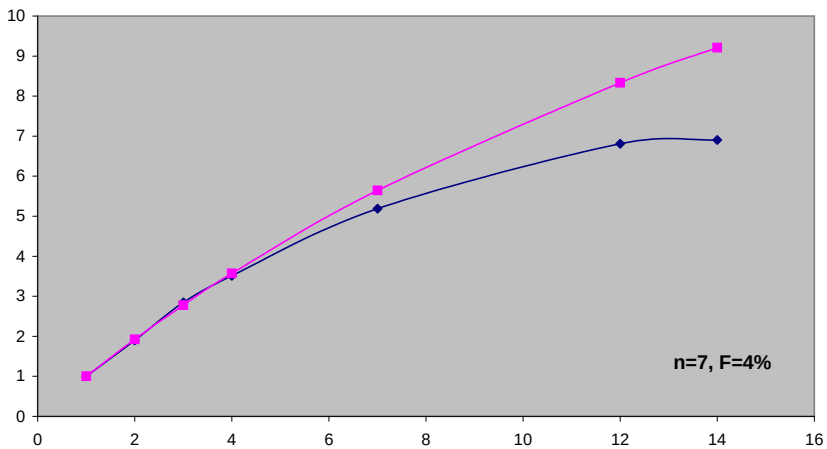






Nonlin. Sol.: Speedup vs. Amdahl's Law

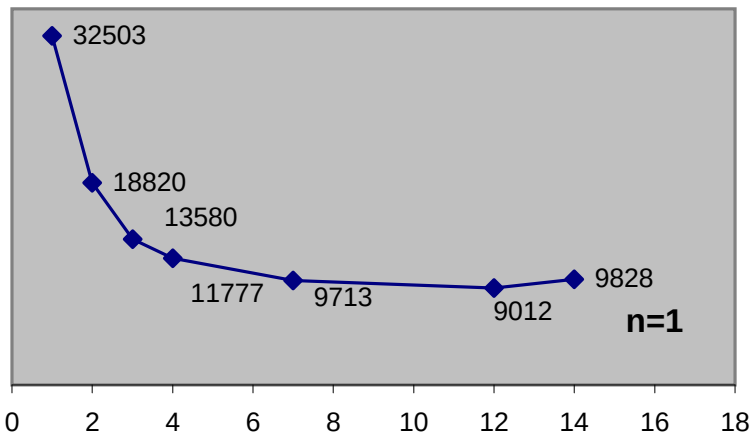


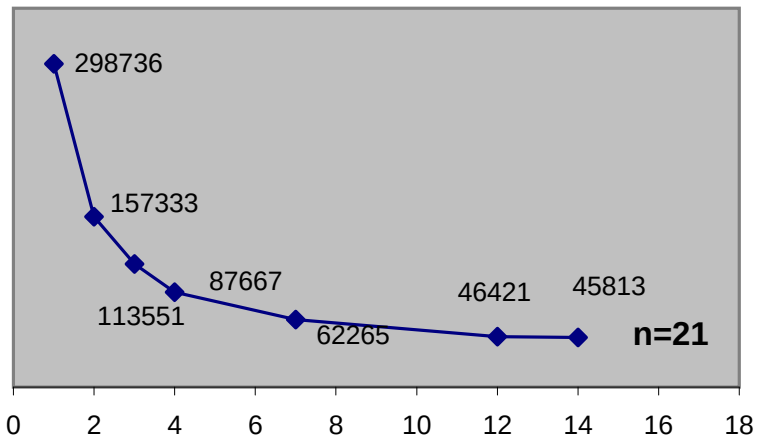


Nonlinear Estimation Experiment

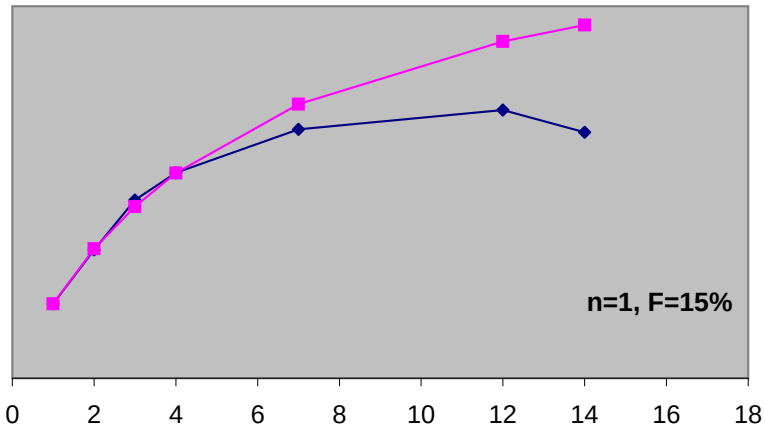
- simulate measurements from the solution specified before
- generate 200 MH draws (starting from almost true values)
- use UKF Filter
- 32 parallel sequences
- for models $N = 1, N = 2$
- set thread pool size to 1, 2, 3, 4, 7, 12, 14
- all experiments use same seed

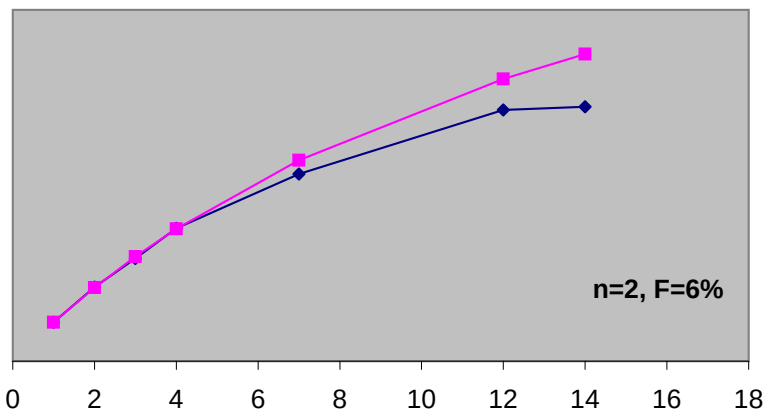
Nonlin. Est.: Computing Times vs. # Processors





Nonlin. Est.: Speedup vs. Amdahl's Law





Conclusion

- we managed to find parallel algorithms that can provide a significant speedup on multi-core computers
- it scales well up to 4-8 processors, after that the speedup decreases more quickly than the theoretical bound
- as expected, the speedup is larger in bigger models
- perfect for dual/quad-core desktops/laptops
- no penalty in sequential use
- runs everywhere without any additional configuration
- the number of processors can be adjusted during runtime due to flexible thread pool pattern

Merci de votre attention.

jbendge.sourceforge.net