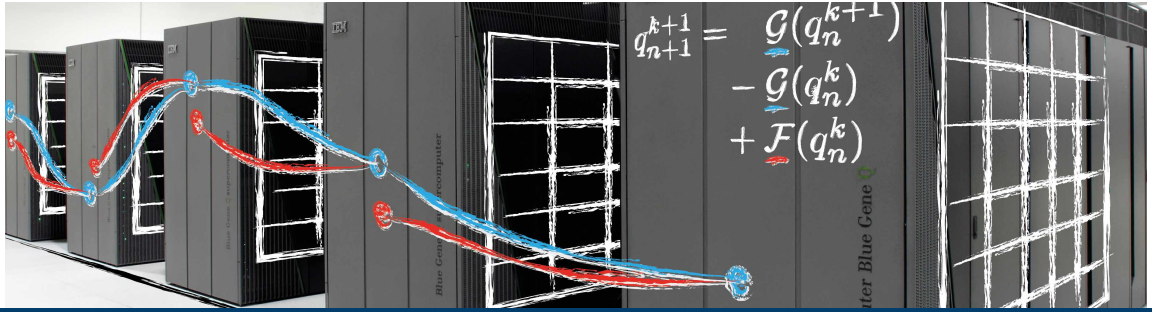


Spectral Deferred Correction

January 30, 2024 | Thomas Baumann | Jülich Supercomputing Centre

Collocation Methods

Want to solve initial value problem in integral form:

$$[u_t(t) = f(t,u),\, u(t_0) = u_0] \iff \left[u(t) = u_0 + \int_{t_0}^{t} f(s,u)ds\right]$$

Discretize integral with quadrature rule:

- Discretize $[t_0, t_0 + \Delta t]$ at M quadrature nodes τ_m: $t_0 \leq \tau_m \leq t_0 + \Delta t$
- Approximate f by polynomial interpolation:

$$f(t,u) \approx \sum_{j=1}^{M} f(\tau_j, u(\tau_j))\, l_j^\tau(t)$$

using Lagrange polynomials

$$l_j^\tau(t) = \frac{\prod_{k=1,k\neq j}^{M}(t - \tau_k)}{\prod_{k=1,k\neq j}^{M}(\tau_j - \tau_k)} \quad \text{with} \quad l_j^\tau(\tau_i) = \delta_{ij}$$

Collocation Methods

Want to solve initial value problem in integral form:

$$[u_t(t) = f(t,u),\, u(t_0) = u_0] \iff \left[u(t) = u_0 + \int_{t_0}^{t} f(s,u)ds\right]$$

Discretize integral with quadrature rule:

- Discretize $[t_0, t_0 + \Delta t]$ at M quadrature nodes τ_m: $t_0 \leq \tau_m \leq t_0 + \Delta t$
- Approximate f by polynomial interpolation:

$$f(t,u) \approx \sum_{j=1}^{M} f(\tau_j, u(\tau_j))\, l_j^{\tau}(t)$$

using Lagrange polynomials

$$l_j^{\tau}(t) = \frac{\prod_{k=1,k\neq j}^{M}(t - \tau_k)}{\prod_{k=1,k\neq j}^{M}(\tau_j - \tau_k)} \quad \text{with} \quad l_j^{\tau}(\tau_i) = \delta_{ij}$$

Collocation Methods

Want to solve initial value problem in integral form:

$$[u_t(t) = f(t,u),\, u(t_0) = u_0] \iff \left[u(t) = u_0 + \int_{t_0}^{t} f(s,u)ds\right]$$

Discretize integral with quadrature rule:

- Discretize $[t_0, t_0 + \Delta t]$ at M quadrature nodes τ_m: $t_0 \leq \tau_m \leq t_0 + \Delta t$
- Approximate f by polynomial interpolation:

$$f(t,u) \approx \sum_{j=1}^{M} f(\tau_j, u(\tau_j))\, l_j^\tau(t)$$

using Lagrange polynomials

$$l_j^\tau(t) = \frac{\Pi_{k=1,k\neq j}^{M}(t - \tau_k)}{\Pi_{k=1,k\neq j}^{M}(\tau_j - \tau_k)} \quad \text{with} \quad l_j^\tau(\tau_i) = \delta_{ij}$$

Collocation Methods Continued

- Recall polynomial approximation: $f(t,u) \approx \sum_{j=1}^{M} f(\tau_j,u(\tau_j))\, l_j^\tau(t)$
- Plug into continuous equation:

$$u(\tau_m) = u_0 + \int_{t_0}^{\tau_m} f(s,u)ds \approx u_0 + \int_{t_0}^{\tau_m} \sum_{j=1}^{M} f(\tau_j,u(\tau_j))\, l_j^\tau(s)ds \tag{1}$$

$$= u_0 + \sum_{j=1}^{M} f(\tau_j,u(\tau_j)) \int_{t_0}^{\tau_m} l_j^\tau(s)ds = u_0 + \sum_{j=1}^{M} q_{m,j} f(\tau_j,u(\tau_j)) \tag{2}$$

- Use quadrature rule Q from integrating Lagrange polynomials to approximate the integral!

Collocation Methods Continued

Use vector notation and rescale quadrature nodes from 0 to 1:

$$(\vec{u})_m = u_m \approx u(\tau_m),\ (\vec{\tau})_m = \tau_m,\ (\vec{u}_0)_m = u_0,\ (Q)_{m,j} = q_{m,j},\ (f(\vec{u}))_m = f(\tau_m, u_m)$$

$$\vec{u} = \vec{u}_0 + \Delta t Q f(\vec{u})$$

Recap:

- Approximate right-hand side by a degree M polynomial
- Use quadrature rule to integrate the polynomial exactly
- For special $\vec{\tau}$, the solution at $t + \Delta t$ has up to order $2M$
- Corresponds to fully implicit Runge-Kutta method, Butcher matrix Q
- Problem: Q is dense $\implies$ direct solve is very expensive!

Collocation Methods Continued

Use vector notation and rescale quadrature nodes from 0 to 1:

$$(\vec{u})_m = u_m \approx u(\tau_m),\ (\vec{\tau})_m = \tau_m,\ (\vec{u}_0)_m = u_0,\ (Q)_{m,j} = q_{m,j},\ (f(\vec{u}))_m = f(\tau_m, u_m)$$

$$\vec{u} = \vec{u}_0 + \Delta t Q f(\vec{u})$$

Recap:

- Approximate right-hand side by a degree M polynomial
- Use quadrature rule to integrate the polynomial exactly
- For special $\vec{\tau}$, the solution at $t + \Delta t$ has up to order $2M$
- Corresponds to fully implicit Runge-Kutta method, Butcher matrix Q
- Problem: Q is dense $\implies$ direct solve is very expensive!

Spectral Deferred Correction

Dutt, Greengard and Rokhlin, 2000

Basic idea

- Use spectral quadrature rule to get solutions of order $2M$ or $2M-1$ (or $2M-2$)
- Solve equation for the error with simple quadrature rule (originally Euler) and refine the solution
- Iterate

Key innovation: Apply deferred corrections to integral form of initial value problem

Error Equation

- Error at iteration k depends on unknown exact solution:

$$\delta^k(t) = u(t) - \vec{u}^k \vec{l}^\tau(t)$$

- Plugging error into initial value problem gives:

$$\delta^k(t) - \int_0^t \left(f\left(\vec{u}^k \vec{l}^\tau(s) + \delta^k(s)\right) - f\left(\vec{u}^k \vec{l}^\tau(s)\right)\right) ds = r^k(t)$$

- Residual depends only on available quantities:

$$r^k(t) = u_0 + \int_0^t f(\vec{u}^k \vec{l}^\tau(s)) ds - \vec{u}^k \vec{l}^\tau(t)$$

- Discretize error equation with "some" quadrature rule Q_Δ at the same nodes τ:

$$\vec{\delta}^k - \Delta t Q_\Delta \left(f\left(\vec{u}^k + \vec{\delta}^k\right) - f\left(\vec{u}^k\right)\right) = \vec{r}^k$$

- Solve this for $\vec{\delta}^k$ and update the solution:

$$\vec{u}^{k+1} = \vec{u}^k + \vec{\delta}^k$$

Error Equation

- Error at iteration k depends on unknown exact solution:

$$\delta^k(t) = u(t) - \vec{u}^k \vec{l}^\tau(t)$$

- Plugging error into initial value problem gives:

$$\delta^k(t) - \int_0^t \left(f\left(\vec{u}^k \vec{l}^\tau(s) + \delta^k(s)\right) - f\left(\vec{u}^k \vec{l}^\tau(s)\right)\right) ds = r^k(t)$$

- Residual depends only on available quantities:

$$r^k(t) = u_0 + \int_0^t f(\vec{u}^k \vec{l}^\tau(s)) ds - \vec{u}^k \vec{l}^\tau(t)$$

- Discretize error equation with "some" quadrature rule Q_Δ at the same nodes τ:

$$\vec{\delta}^k - \Delta t Q_\Delta \left(f\left(\vec{u}^k + \vec{\delta}^k\right) - f\left(\vec{u}^k\right)\right) = \vec{r}^k$$

- Solve this for $\vec{\delta}^k$ and update the solution:

$$\vec{u}^{k+1} = \vec{u}^k + \vec{\delta}^k$$

Error Equation Continued

What have we gained?
$\rightarrow$ Nothing, if we solve the error equation with the same $Q_\Delta = Q$ we used for the collocation problem!

Need simpler quadrature rule Q_Δ (called preconditioner) to solve for the error.
For instance, implicit Euler:

$$Q_\Delta = \begin{pmatrix} \tau_2 - \tau_1 & 0 & 0 & \dots & 0 \\ \tau_2 - \tau_1 & \tau_3 - \tau_2 & 0 & \dots & 0 \\ \vdots & \vdots & \ddots & \dots & 0 \\ \tau_2 - \tau_1 & \tau_3 - \tau_2 & \dots & \dots & \tau_M - \tau_{M-1} \end{pmatrix}$$

Error Equation Continued

Resulting iteration after algebraic manipulation:

$$(I - \Delta t Q_\Delta f)(\vec{u}^{k+1}) = \vec{u}_0 + \Delta t (Q - Q_\Delta) f(\vec{u}^k)$$

Compare to vanilla implicit Euler:

$$(1 - \Delta t f)(u) = u_0$$

Now did we gain something?

- We better choose Q_Δ lower triangular such we can solve with forward substitution
- We just need to solve implicit Euler steps with modified step size and right-hand side
- If we choose Q_Δ diagonal, we can solve for the nodes in parallel!
- Consider PDE with N degrees of freedom: Collocation problem is size $NM \times NM$, but if SDC convergences after K iterations, it requires KM solves of $N \times N$ systems
- Typically, gain one order of accuracy per iteration, dependent on Q_Δ and problem

Modern Interpretation of SDC

- Consider fully implicit collocation problem: $(I - \Delta t Q f)(\vec{u}) = \vec{u}_0$
- Simplest iterative approach: Picard iteration:

$$\vec{u}^{k+1} = \vec{u}^k - \underbrace{\left((I - \Delta t Q f)(\vec{u}^k) - \vec{u}_0\right)}_{\vec{r}^k}$$

  $\rightarrow$ poor stability because it is explicit
- Precondition the Picard iteration with Q_Δ:

$$(I - \Delta t Q_\Delta f)\,(\vec{u}^{k+1}) = \vec{u}_0 + \Delta t\,(Q - Q_\Delta)\,f(\vec{u}^k)$$

- Looks familiar! SDC = preconditioned Picard iteration

Modern Interpretation of SDC Continued

Construct SDC iteration matrix

- Consider linear test equation: $u_t = \lambda u$
- SDC iteration becomes:

$$\vec{u}^{k+1} = \underbrace{(I - \Delta t Q_\Delta \lambda)^{-1} \Delta t \, (Q - Q_\Delta) \, \lambda \vec{u}^k}_{G\vec{u}^k} + \underbrace{(I - \Delta t Q_\Delta \lambda)^{-1} \vec{u}_0}_{c}$$

- Error behaves as $\vec{e}^{k+1} = G\vec{e}^k$
- Convergence:
  - Look for Q_Δ with $\rho(G) < 1$
  - Look for Q_Δ with $||G|| < 1$
  - Look for Q_Δ that make G nilpotent

Q_Δ is now a preconditioner and not necessarily a quadrature rule!

Modern Interpretation of SDC Continued

Look at stiff limit of SDC iteration matrix

Stiff limit $\lambda \to -\infty$, $\lambda \in \mathbb{R}$:

$$\vec{e}^{k+1} \approx \underbrace{\left(I - Q_\Delta^{-1} Q\right)}_{G} \vec{e}^{k}$$

- LU: $Q_\Delta = U^T$ with $LU = Q^T$ and $L_{ii} = 1$

  $$G = I - \left(U^T\right)^{-1} U^T L^T = I - L^T$$

  is strictly upper triangular and hence nilpotent (Weiser, 2015)

- MIN: Numerically minimize spectral radius of G with Q_Δ diagonal (Speck, 2017)

Why bother with SDC?

SDC is closely related to Runge-Kutta

- Converged collocation problem solution is solution to fully implicit Runge-Kutta method
- SDC with fixed number of iterations is a Runge-Kutta method
  (Keep an eye out for work by Fregin and Bronasco)
- For non-stiff problems, explicit Runge-Kutta methods are very hard to beat with SDC

but...

- Special time-marching schemes easier to construct in low order, use SDC to get higher order
- Can accelerate SDC with inexactness, adaptive resolution between iterations, ...
- Parallel-in-Time (PinT) extensions

$\rightarrow$ Much greater flexibility than most other RK schemes

Example of SDC flexibility: Implicit-Explicit splitting

Ruprecht and Speck, 2016

- Replace $Q_\Delta f$ with $Q_{\Delta,I} f_I + Q_{\Delta,E} f_E$ in SDC iteration
- Choose $Q_{\Delta,E}$ strictly lower triangular for explicit integration
- Done

Parallel-in-Time extensions

Parallelization across the method

Compute all M stages concurrently via diagonal Q_Δ

See recent work by Caklovic, Lunet, Götschel and Ruprecht (2024):

- MIN-SR-NS: Nilpotent in the non-stiff limit
- MIN-SR-S: Numerically minimize $\rho(G)$ in the stiff limit, but good!
- MIN-SR-FLEX: Nilpotent in the stiff limit, but Q_Δ changes between iterations

Parallel-in-Time extensions

Parallelization across the steps

Start by assembling composite collocation problem by gluing together L steps with transfer operator N

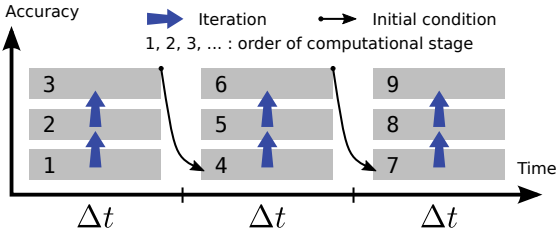
$$\begin{pmatrix} I - \Delta t Q F & & & \\ -N & I - \Delta t Q F & & \\ & \ddots & \ddots & \\ & & -N & I - \Delta t Q F \end{pmatrix} \begin{pmatrix} \vec{u}_1 \\ \vec{u}_2 \\ \vdots \\ \vec{u}_L \end{pmatrix} = \begin{pmatrix} \vec{u}_0 \\ 0 \\ \vdots \\ 0 \end{pmatrix}$$

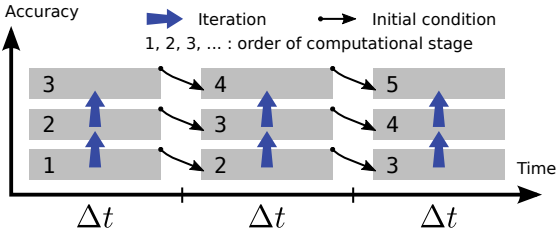
Then solve in parallel using, for instance,

- Pipelining, i.e. iterate block Gauß-Seidel (Guibert and Tromeur-Dervout, 2007)
- PFASST (Emmett and Minion, 2012)

Parallel-in-Time extensions

Block Gauß-Seidel pipelining





Start iteration on step as soon as one iteration has been performed on previous step

Figures by Thibaut Lunet

Step size adaptivity in SDC

T.B., Lunet, Götschel, Ruprecht, Speck (2024)

Transfer ideas from embedded RKM

- Use same step size update equation
- Use bespoke error estimates

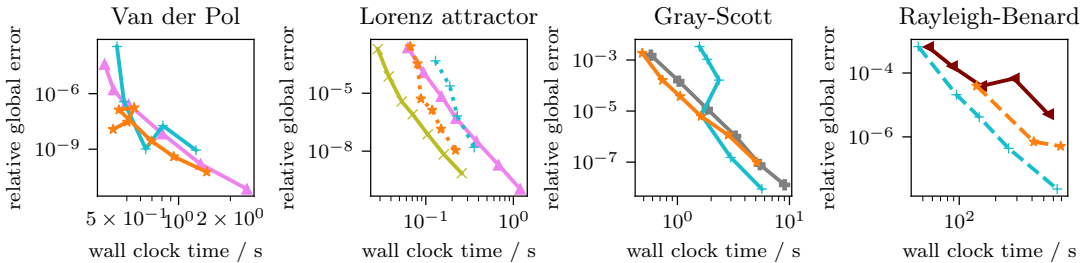
Algorithm 1: Δt-adaptivity

- Constant number of iterations, adaptive step size
- Error estimate based on getting order k after k iterations

Algorithm 2: Δt-k-adaptivity

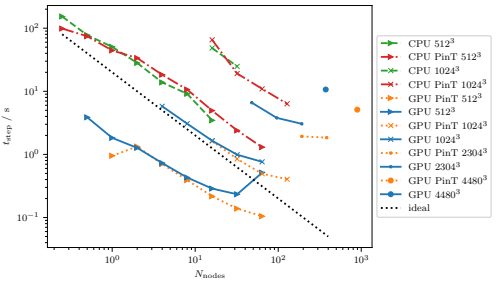
- Choose both Δt and k adaptively
- Error estimate based on polynomial interpolation defined by converged collocation problem

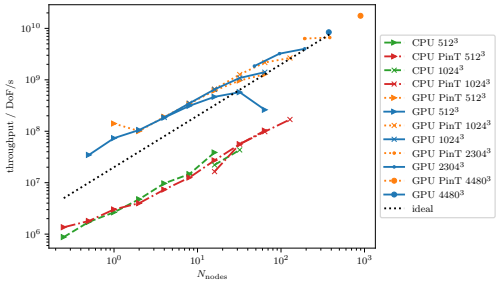
Time-parallel adaptive SDC vs. embedded RKM



- At least mode of adaptive PinT SDC is always competitive with RKM for stiff problems
- In Rayleigh-Benard, no high order comparison RKM available, SDC still better at order 3

Parallel scaling of Gray-Scott implementation



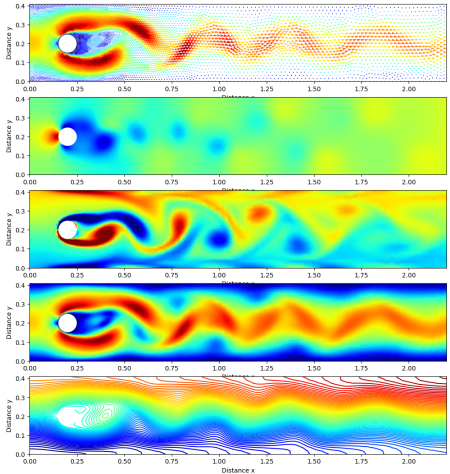


Use diagonal SDC to extend scaling

- Shifts communication from all-to-all to reduce in this spectral discretization
- Improves strong scaling
- Enables scaling up to 3584 GPUs

Parallel SDC for Navier-Stokes equations

Monolithic SDC with diagonal preconditioners (Abdelouahed Ouardghi)



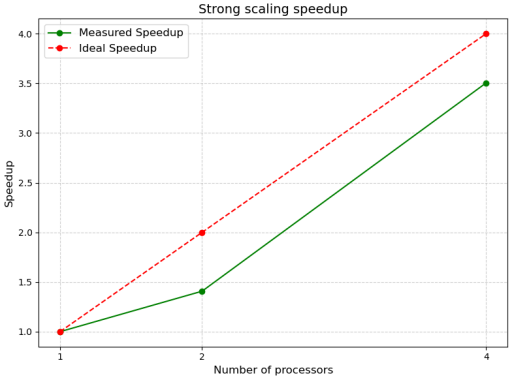
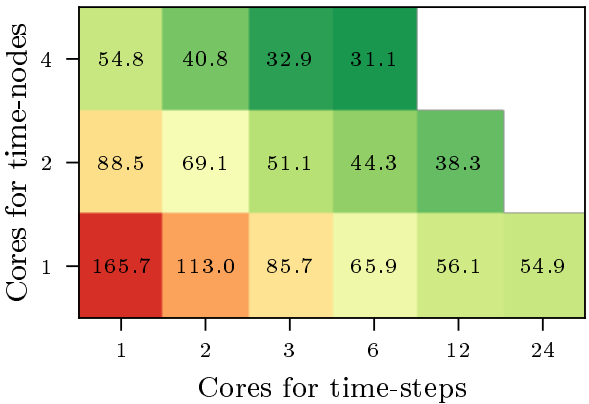


Figure: Left: Flow around the cylinder, DFG95 benchmark. Right: Speedup with diagonal SDC

PFASST-ER: PFASST + diagonal SDC

Schöbel, Speck (2019)

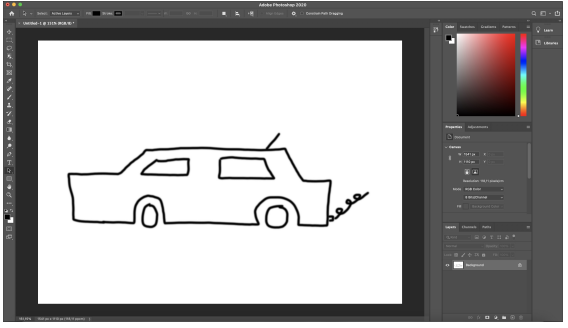
Idea: Use parallel SDC sweeps within parallel time-steps
Example: 2D Allen-Cahn, fully-implicit, 256x256 DOFs in space, up to 24 available cores.



<commercial>

pySDC - Prototyping Spectral Deferred Corrections

Test before you invest at `https://parallel-in-time.org/pySDC`



pySDC - Prototyping Spectral Deferred Corrections

Test before you invest at `https://parallel-in-time.org/pySDC`

Tutorials and examples

- Ships with a lot of examples
- Many SDC flavors up to PFASST
- Problems beyond heat equation

Parallel and serial

- Serial algorithms
- Pseudo-parallel algorithms
- Time-parallel algorithms
- Space-time parallel algorithms

Python

- Interface compiled code for expensive spatial solves
- Implementation close to formulas

CI/CD/CT

- Well documented
- Well tested
- Works ~~on my machine~~ anywhere
- Reproduce paper results

JÜLICH
Forschungszentrum

Code separated into modules

Problem

- implicit Euler like solves
- evaluate right hand side
- initial conditions, maybe exact solution
- use your own datatype

Sweeper: Timestepping

- assembles and calls solves in problem class
- administers right hand side evaluations
- takes care of Q_Δ, splitting etc.
- DIRK methods available as sweepers

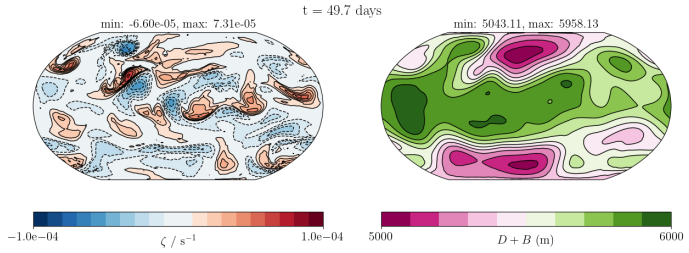
Callbacks: Modify anything at any time

- solution
- step size
- sweeper
- ...

Hooks: Extract anything at any time

- Newton / SDC iterations and f evaluations
- wall time
- error
- ...

pySDC is now compatible with Firedrake and Gusto!



SCAN ME

How to use

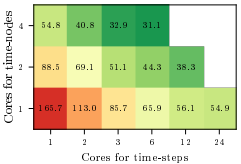
- Setup custom problem class using Firedrake
- Setup any SDC scheme in pySDC and use as Gusto time discretization
- Works with space-time parallel simulations
- See tutorials, step 7 on the pySDC github page (scan QR code)

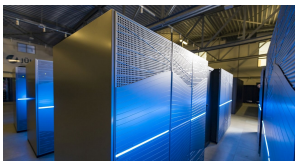
</commercial>

Three takeaways

Spectral Deferred Corrections (SDC) are a great playground for research on time integration methods

Lots of SDC variants and their combination can lead to highly competitive time integration methods





Prototyping ideas, with real code, on real (parallel) machines, is crucial to find out about potential and limitations