SeASiTe

Selbstadaption für zeitschrittbasierte Simulationstechniken auf heterogenen HPC-Systemen

Self-Adaptation of Time-Step-based Simulation Techniques on Heterogeneous HPC Systems

Projektpartner: Universität Bayreuth, TU Chemnitz, FAU Erlangen-Nürnberg, Megware (assoziiert)

Thomas Rauber, Universität Bayreuth

7. HPC-Status-Konferenz der Gauß-Allianz
4.-5. Dezember 2017, HLRS Stuttgart

Team des Projektes SeASiTe

- **Universität Bayreuth:**
  Thomas Rauber, Matthias Korch,
  Johannes Seiferth, Markus Straubinger
- **Technische Universität Chemnitz:**
  Gudula Rünger,
  Michael Hofmann, Robert Kiesel, Marcel Richter
- **FAU Erlangen-Nürnberg:**
  Gerhard Wellein,
  Christie Alappat, Thomas Röhl
- **Megware (assoziiert):**
  Axel Auweter, Jürgen Gretzschel

Outline

Ausgangsfragestellung

- Zunehmende **Heterogenität der Architektur** von HPC-Systemen verlangt nach neuartigen Programmierkonzepten.
  → **Herausforderung:** Ausnutzung der erhöhten Rechenleistung der Architektur der HPC-Systeme
- Zunehmend **komplexe HPC-Simulationsverfahren** zur Simulation immer rechenintensiver Szenarien.
  → **Herausforderung:** großer Programmieraufwand:
  Simulationsprogramme sollen langfristig genutzt werden können.

Problematik: Die Lebensdauer der Simulationsprogramme ist wesentlich länger als die Lebensdauer der HPC-Hardware
→ **Ausgangsfrage:** Wie kann sich Simulationssoftware weitgehend selbständig an die sich ständig wandelnde heterogene HPC-Hardware anpassen?
→ Wie kann **Portabilität der Performance** erreicht werden?

Gesamtziel des Vorhabens

Gesamtziel: Weitgehend selbständige Anpassung der jeweiligen HPC-Software an die Hardwaredetails der heterogenen HPC-Architektur und der Eingabedaten mit Erreichung **hoher Effizienz** (Zeit, Energie).

Ziele im Detail:

- Entwicklung neuer Selbstadaptionstechniken für HPC-Anwendungen auf **heterogenen HPC-Systemen**.
  $\rightarrow$ automatische Anpassung der Programme und damit **performance-optimale Ausführung** von HPC-Anwendungen
- Entwicklung von Selbstadaptionsmethodiken für langlaufende **zeitschrittbasierte HPC-Simulationen**
- **Integrationsmechanismen** zur einfachen Einbettung solcher Selbstadaptionstechniken in HPC-Simulationsprogramme

Forschungsansatz

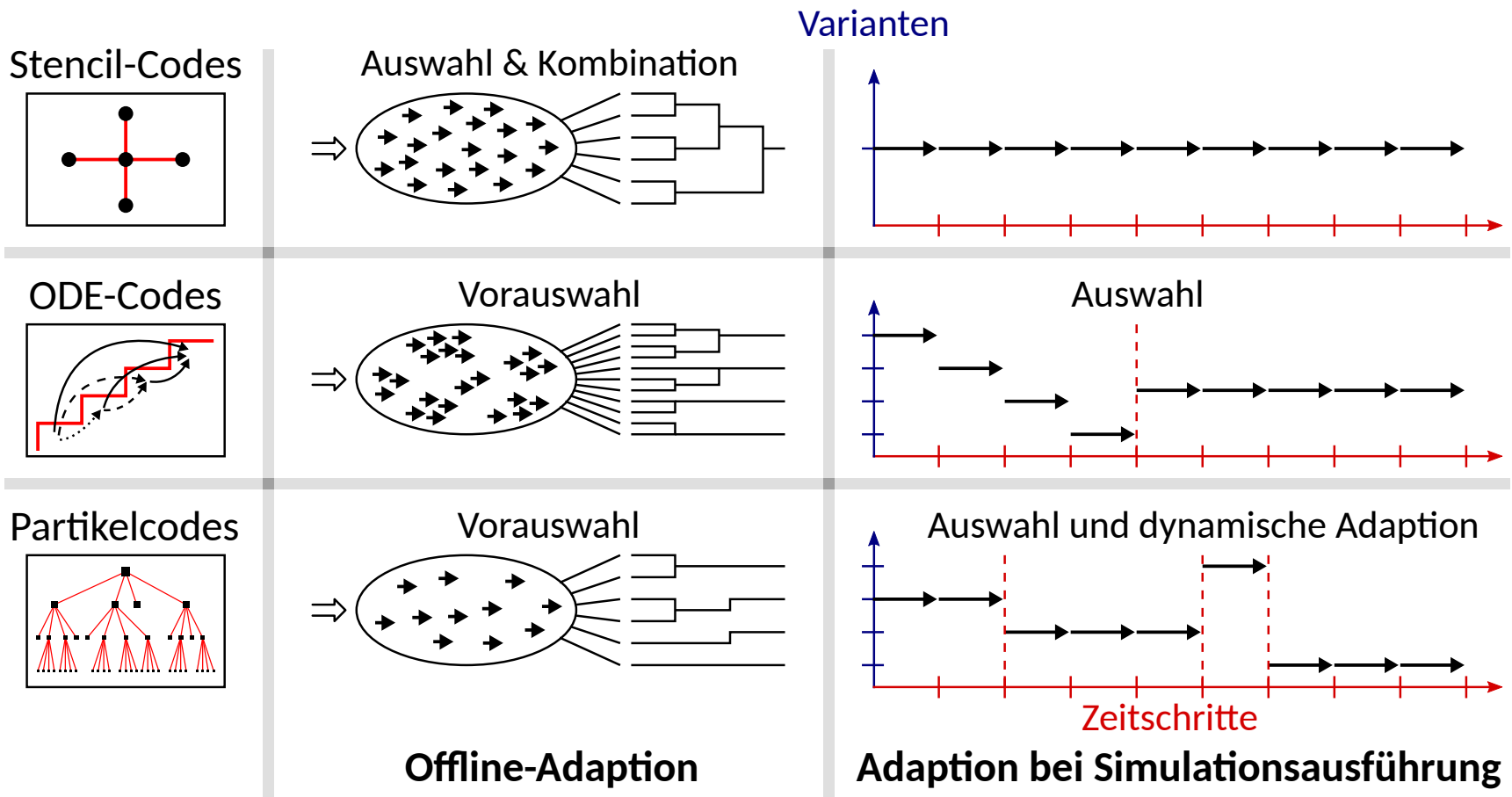
Einbeziehung spezifischer Eigenschaften und des Laufzeitverhaltens der Simulationsalgorithmen in die **Selbstadaption**:

- Unterteilung zeitschrittbasierter Algorithmen in **reguläre und irreguläre Algorithmen**.
  $\rightarrow$ Berücksichtigung unterschiedlicher Klasssen von Algorithmen für die Selbstadaption
- **Offline-Phase** zur Vorauswahl von Programmvarianten auf der Basis eines **Performance-Modells** (ECM-Modell)
  unterschiedliche Optimierungskriterium (Laufzeit, Energie) wählbar
- **Online-Phase**: verwende erste Zeitschritte zum Austesten ausgewählten Programmvarianten mit den gegebenen Eingabedaten
- Verlaufskontrolle mithilfe eines **Hardwareanalysemechanismus** zur Erfassung wesentlicher Änderungen des Laufzeit- oder Energieverhaltens

Geplantes Vorgehen

Betrachtung spezieller **Klassen von Simulationsalgorithmen**

Entwicklung und Integration selbstadaptierender Simulationsverfahren

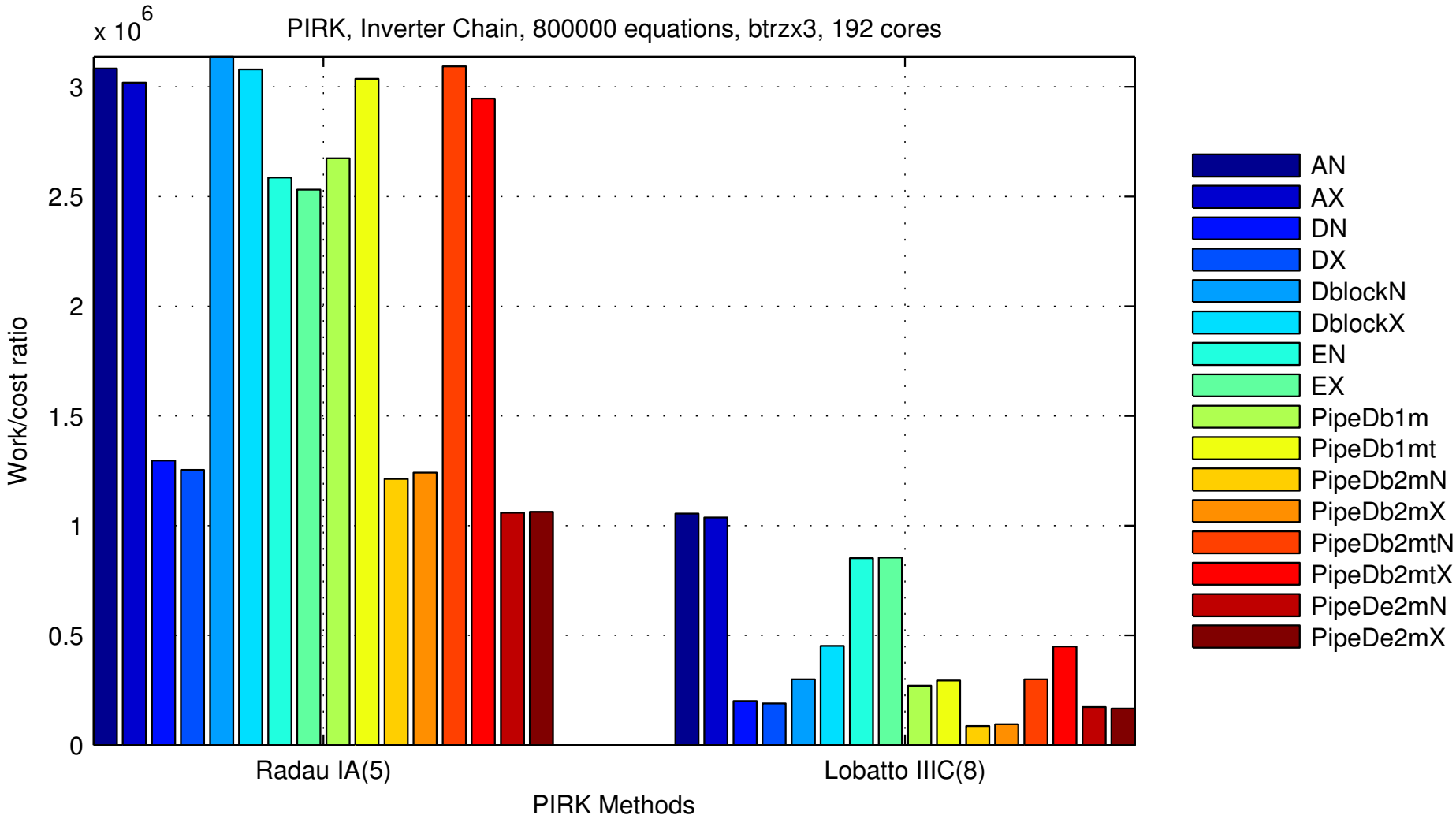


Outline

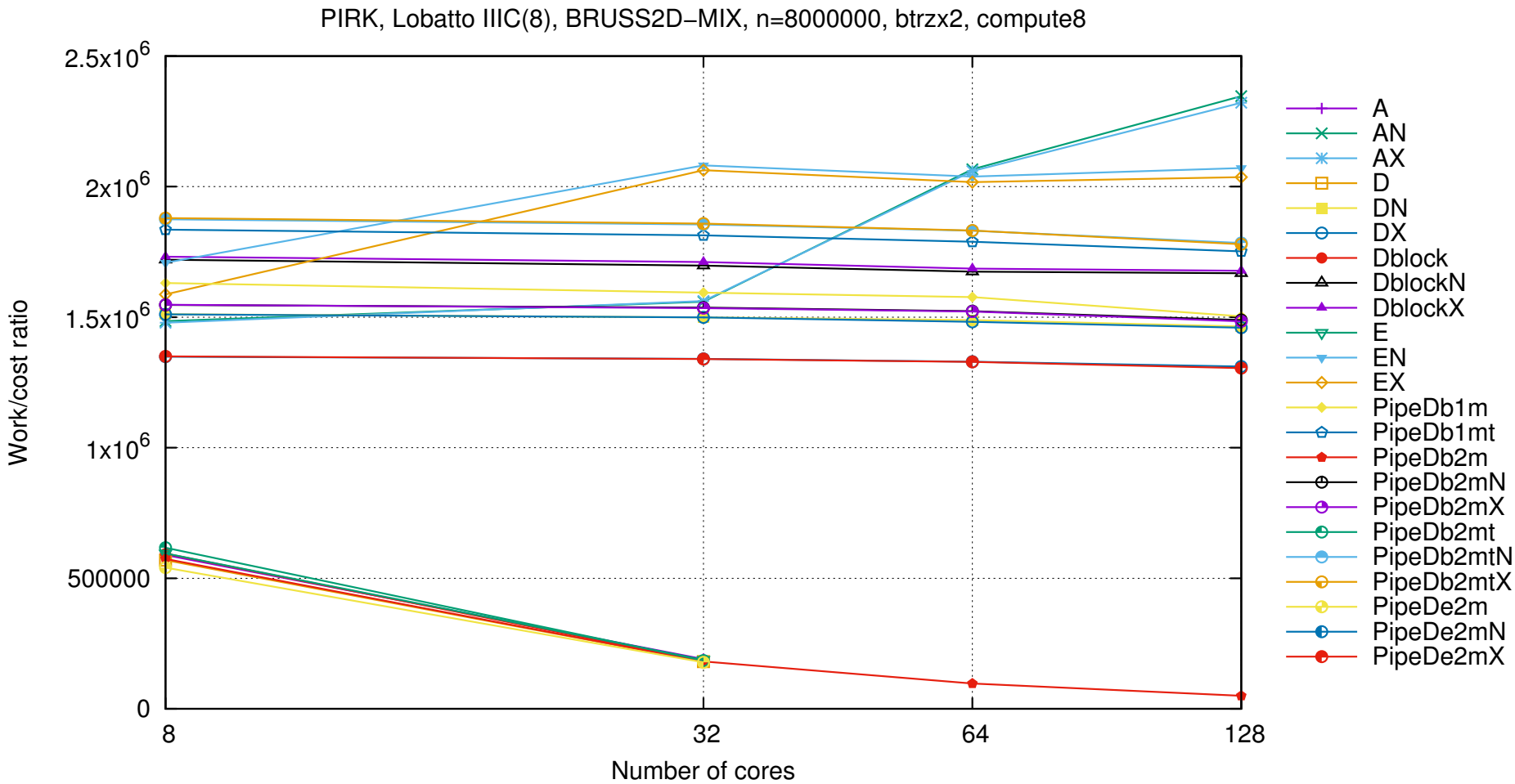
Lösungsverfahren für ODEs

- **Anfangswertprobleme**:
  gegeben ODE $\mathbf{y}'(t) = \mathbf{f}(t, \mathbf{y}(t))$ mit $\mathbf{y}(t_0) = \mathbf{y}_0$.
- numerische Lösung mit **Zeitschrittverfahren**
- **Beispiel:** iterierte Runge-Kutta (RK) Verfahren mit s Stufen:
  in jedem Zeitschritt erfolgen folgende Berechnungen:
  - für $l = 1, \ldots, s$: $\mathbf{Y}_l^{(0)} = \eta_\kappa$, (Initialisierung)
  - für $k = 1, \ldots, m$, $l = 1, \ldots, s$:
    $\mathbf{Y}_l^{(k)} = \eta_\kappa + h_\kappa \cdot \sum_{i=1}^{s} a_{li} \cdot \mathbf{F}_i^{(k-1)}$
    mit $\mathbf{F}_i^{(k-1)} = \mathbf{f}(t_\kappa + c_i h_\kappa, \mathbf{Y}_i^{(k-1)})$
- Berechnung des neuen Approximationsvektors:
  $\eta_{\kappa+1} = \eta_\kappa + h_\kappa \sum_{l=1}^{s} b_l \mathbf{F}_l^{(m)}$
  $\hat{\eta}_{\kappa+1} = \eta_\kappa + h_\kappa \sum_{l=1}^{s} \hat{b}_l \mathbf{F}_l^{(m-1)}$
- Fehler- und Schrittweitenkontrolle mit $\eta_{\kappa+1}$ und $\hat{\eta}_{\kappa+1}$

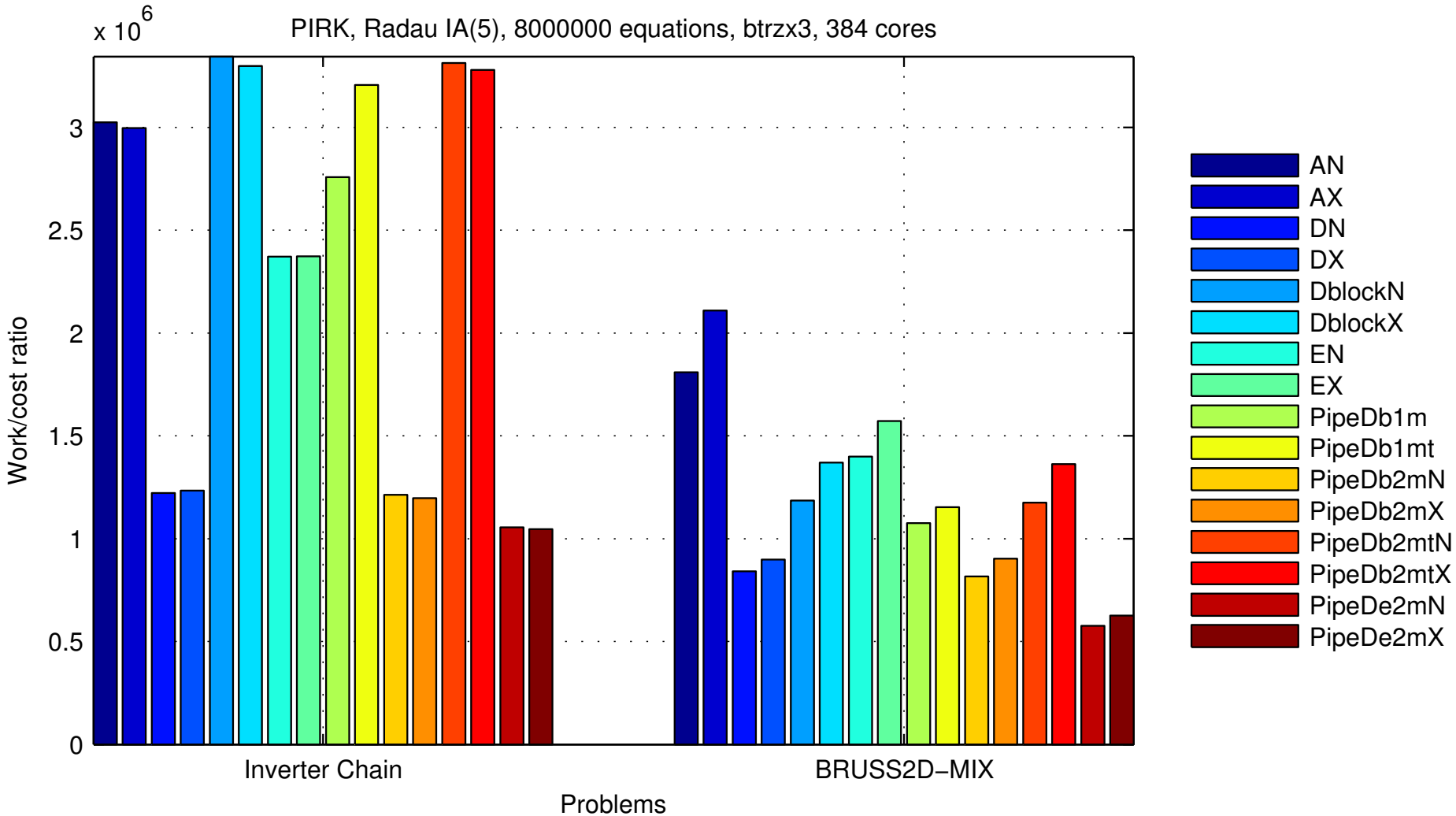
Abhängigkeit vom Basis RK Verfahren



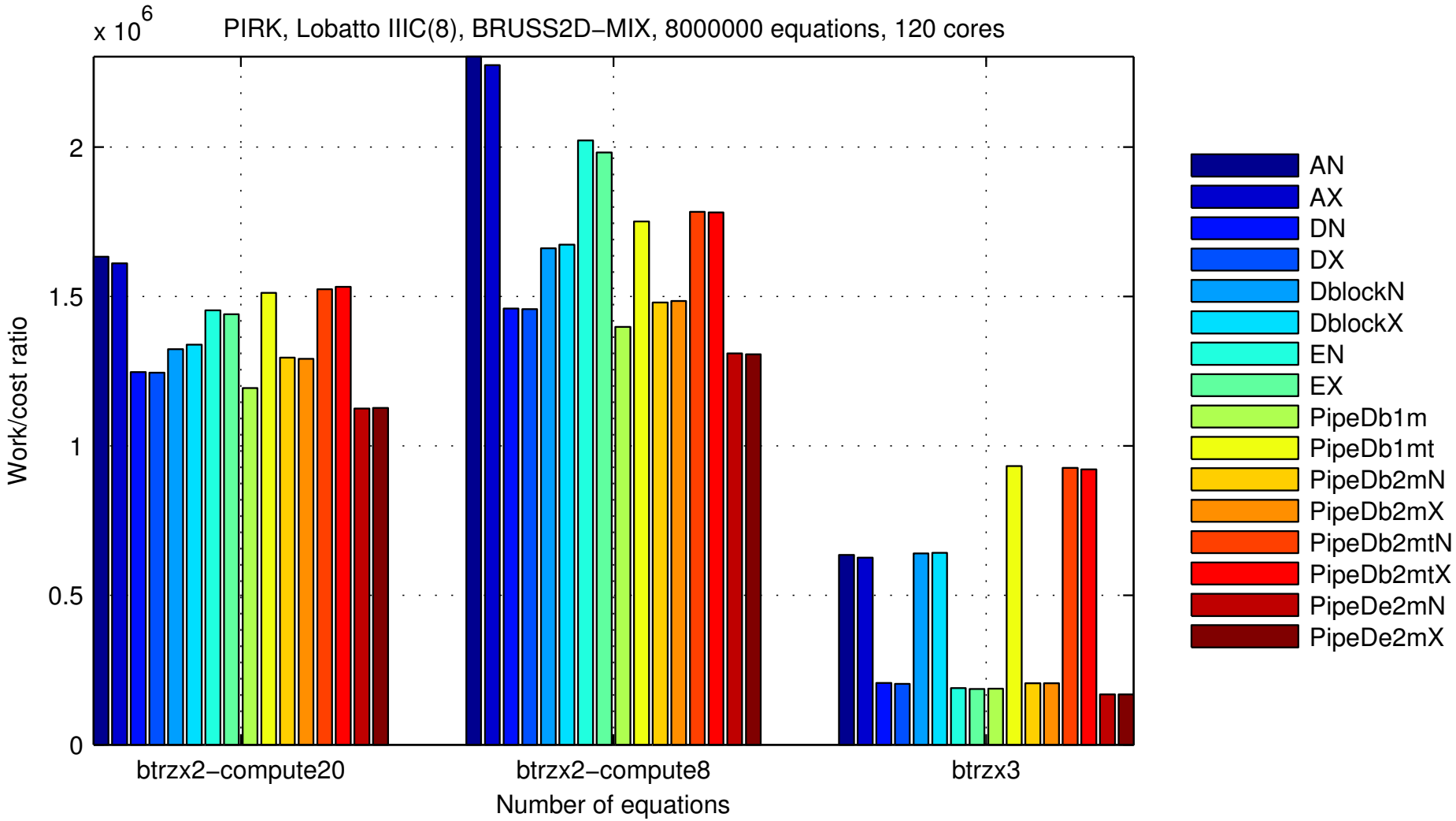
Abhängigkeit von der Anzahl der Kerne



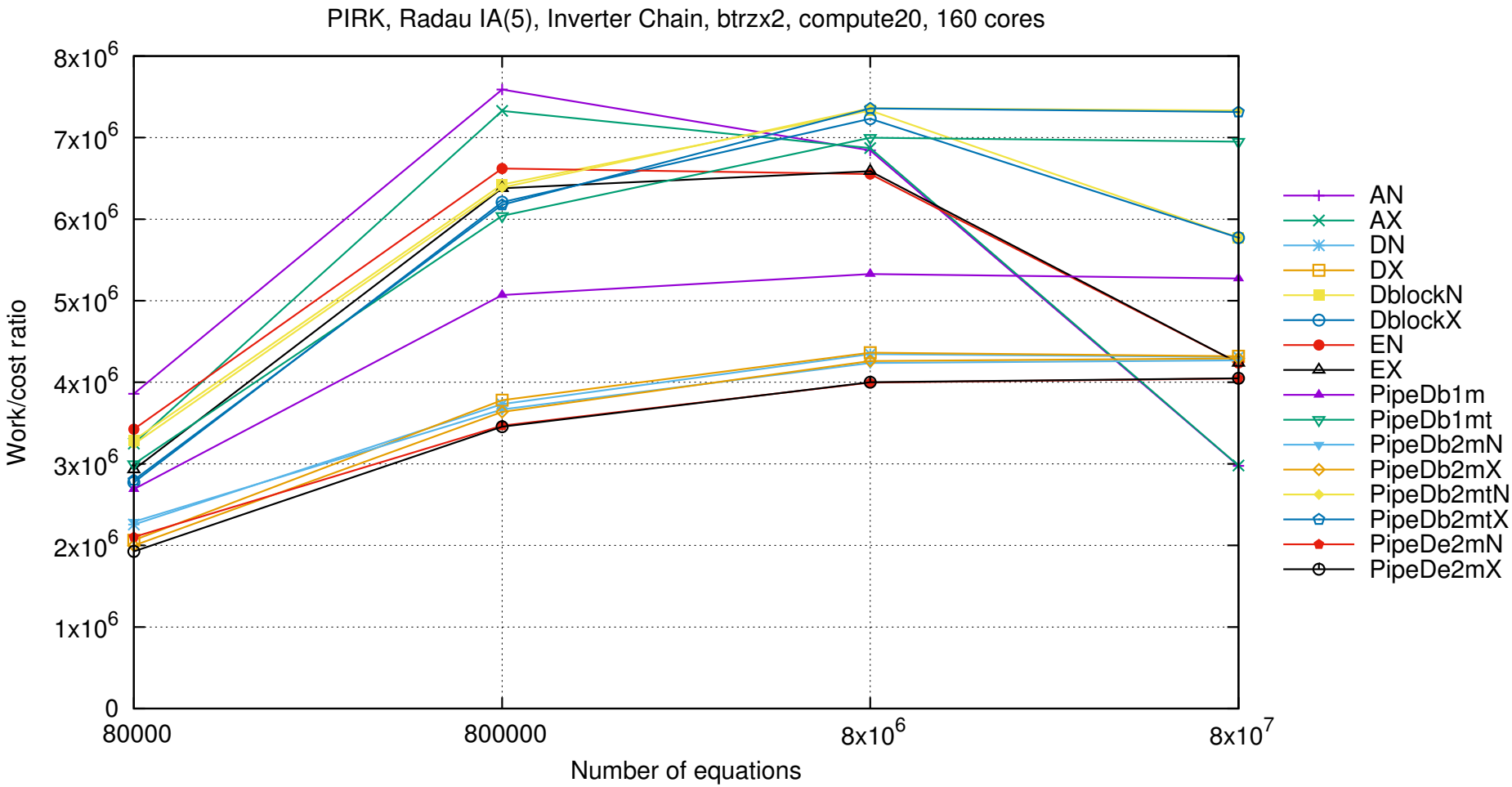
Abhängigkeit von dem zu lösenden ODE Problem



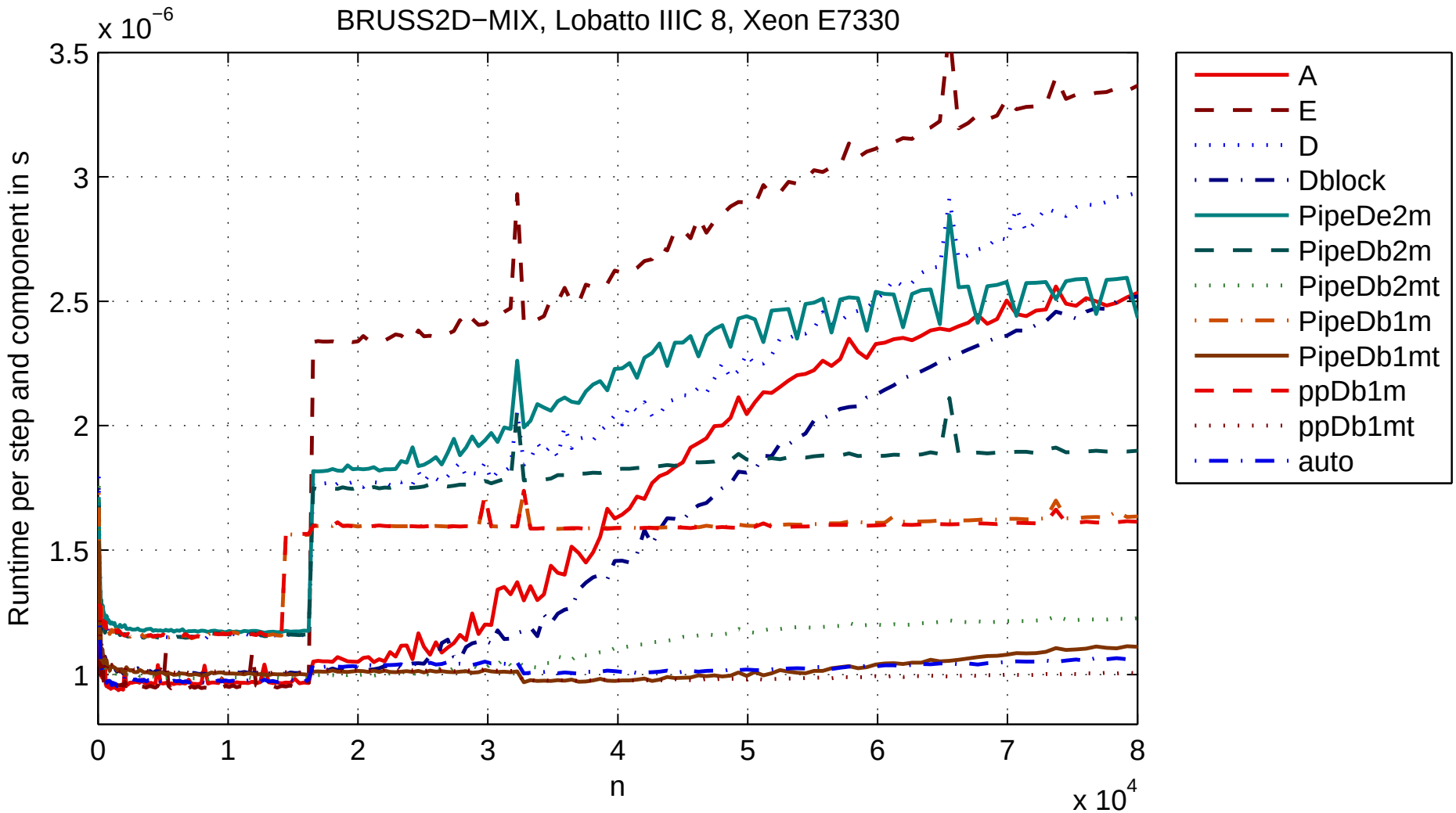
Abhängigkeit von der verwendeten HPC-Plattform



Abhängigkeit von der Problemgröße



Dynamische Variantenauswahl für Iterierte RK Implementatierungen

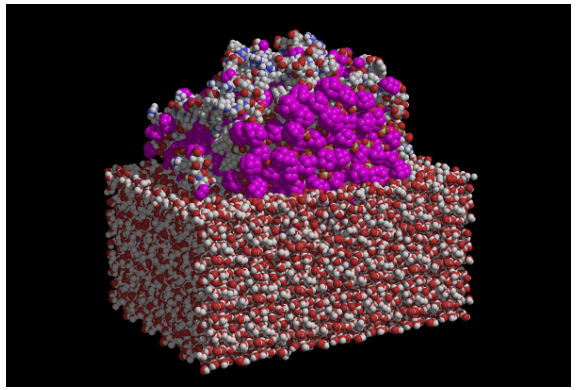


☞ Auswahl der schnellsten Implementierung zur Laufzeit

Outline

Partikelsimulationen – ScaFaCoS

- Als Applikationen im Bereich dynamisch irrgerulärer Verfahren werden langreichweitige Partikelwechselwirkungen betrachtet
- Coloumb Kräfte benötigen die meiste Rechenzeit in jedem Zeitschritt
- Zwei vorherrschende Datenstrukturen in Algorithmen werden exemplarisch berücksichtigt: Baumstrukturen in hierarchischen Methoden und Gitterstrukturen in Fourier-basierten Methoden
- Partikelsimulationen sind zeitschrittbasiert, wodurch zwischen den Schritten Potential für Anpassungen während der Laufzeit besteht
- untersuchte Methoden: P^2NFFT und FMM



ApoA1 in Wasser

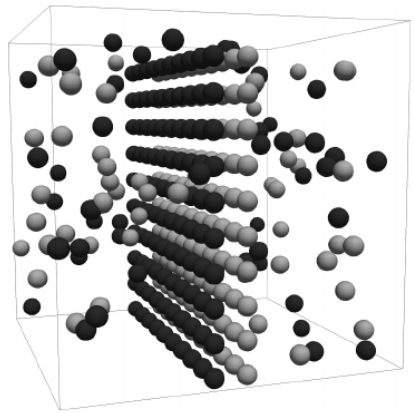
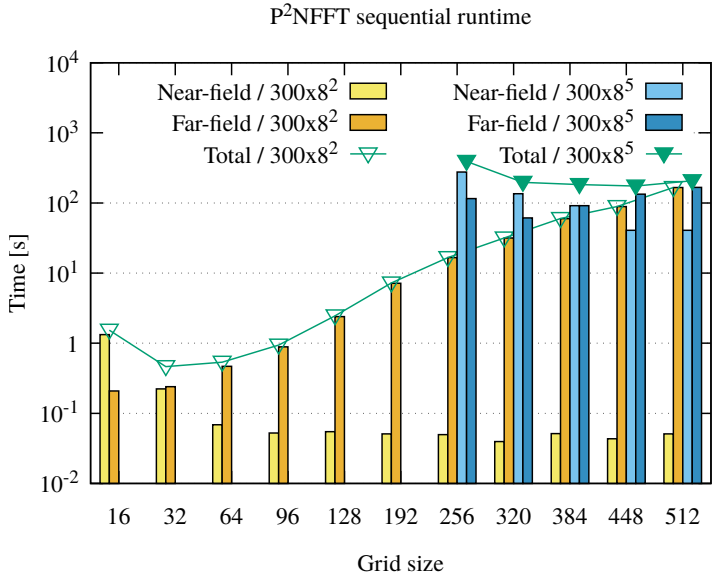


FIG. 4. The cloud-wall system (300 charges): two oppositely charged walls in the center of the box and a surrounding diffuse cloud. The system was artificially created to contain a strong long-range field component.

Quelle: Arnold et al: Comparison of Scalable Fast Methods for Long-Range Interactions,Phys. Rev. E, 88(6), 063308, 2013

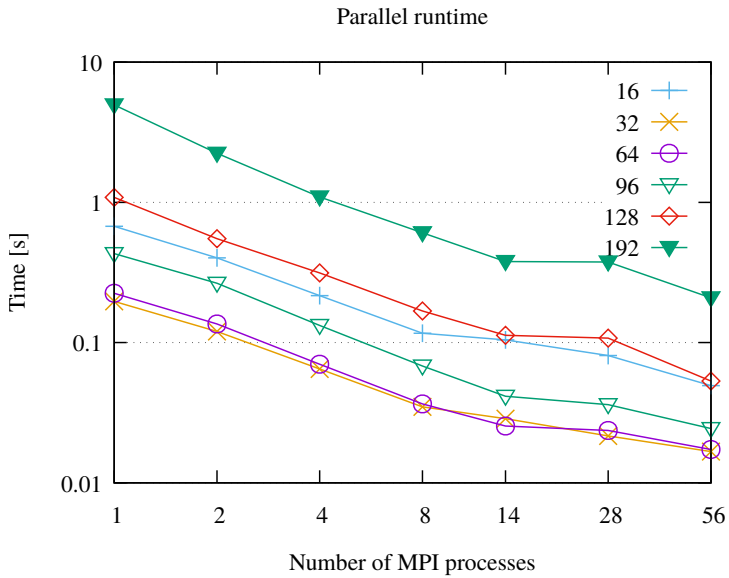
Untersuchungen zu ScaFaCoS - P^{2}NFFT Parameter



- P^{2}NFFT; Particle-Particle Nonequispaced Fast Fourier Transform
- Messungen auf Intel Haswell-System mit 2 x Xeon E5-2683 v3 mit je 14 Kernen und 35.840 KB L3-Cache

- Zwei unterschiedlich große Partikelsysteme untersucht: 300×8^2 und 300×8^5
- Die Abbildung zeigt die sequenzielle Laufzeit der P^{2}NFFT Methode abhängig von der Gittergröße
- P^{2}NFFT ist eine Fourier-basierte Methode, die auf einer Gitterstruktur aufbaut und zwischen Nah- und Fernfeld unterscheidet
- Eine höhere Gittergröße bewirkt einen höheren Aufwand für die Fernfeldberechnung und eine Verkleinerung des Aufwandes der Nahfeldberechnung

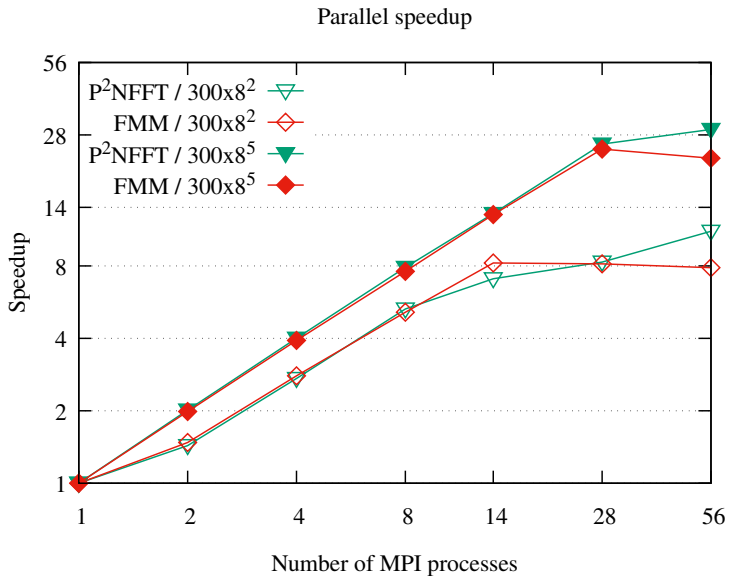
Untersuchungen zu ScaFaCoS - P^{2}NFFT Parallelität



- Messungen mit unterschiedlicher Anzahl von MPI Prozessen
- Verwendung eines Partikelsystems mit 300×8^2 Partikeln

- Untersucht wurden die Parallelisierung von P^{2}NFFT bei verschiedenen Gittergrößen
- Alle Gittergrößen zeigen ein ähnliches Verhalten der Skalierbarkeit
- Gittergrößen 32 und 64 führen zu den geringsten Laufzeiten; andere Gittergrößen führen teilweise zu wesentlich höheren Laufzeiten;
- Die ideale Gittergröße hängt auch von der Verteilung der Partikel ab.

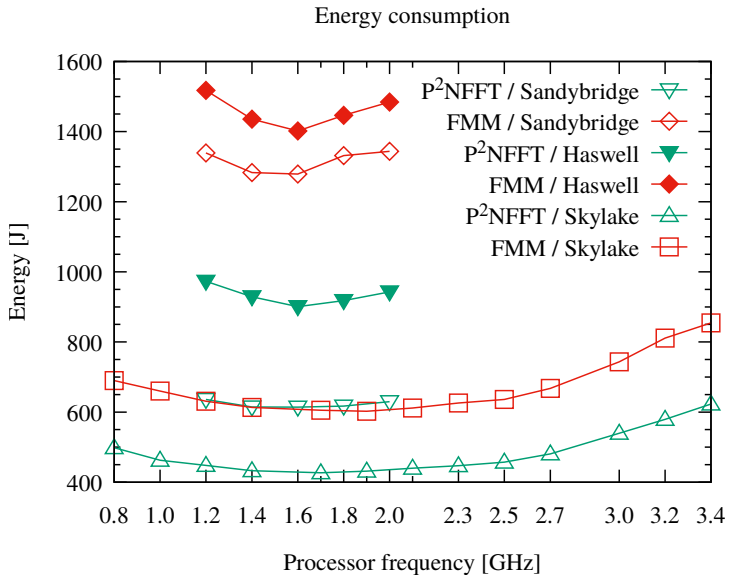
Untersuchungen zu ScaFaCoS - P^{2}NFFT und FMM



- FMM: Fast Multipole Method
- FMM arbeitet im Gegensatz zu P^{2}NFFT auf einer Baumstruktur
- Kontrolle zwischen Nah- und Fernfeldberechnung durch maximale Baumtiefe

- Untersuchte Partikelsysteme: 300×8^2 und 300×8^5
- Es wird für das große Partikelsystem bis 28 Kerne für P^{2}NFFT und FMM ein nahezu idealer Speedup erreicht
- P^{2}NFFT profitiert mehr von Hyperthreading als FMM

Untersuchungen zu ScaFaCoS - Frequenz

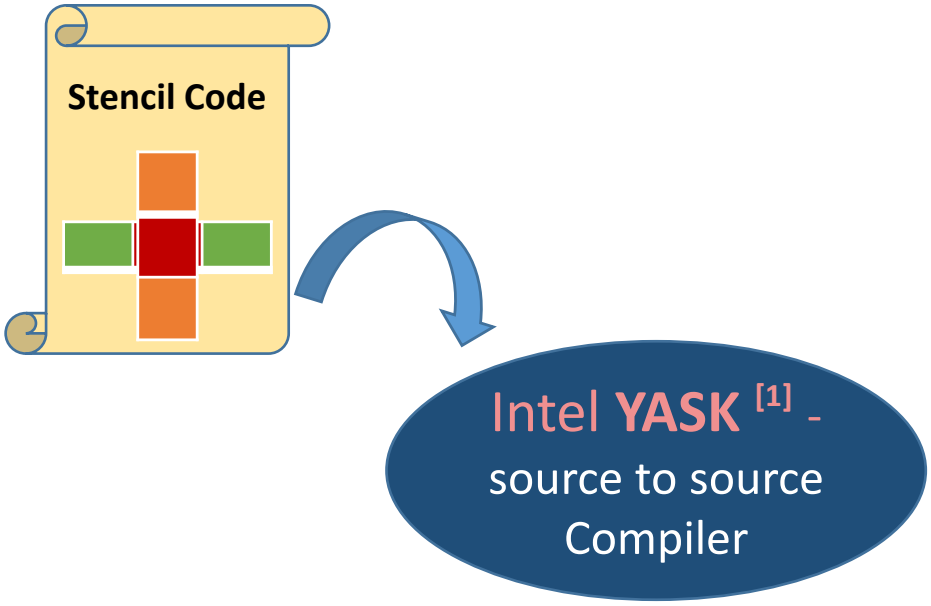


- Messungen fanden auf drei Systemen mit dem Partikelsystem mit 300×8^5 Partikeln statt
- Sandybridge 2 $\times$ Xeon E5-2650 je 8 Kerne, Haswell 2 $\times$ Xeon E5-2683 v3 je 14 Kerne, Skylake Core i7-6700 mit 4 Kernen

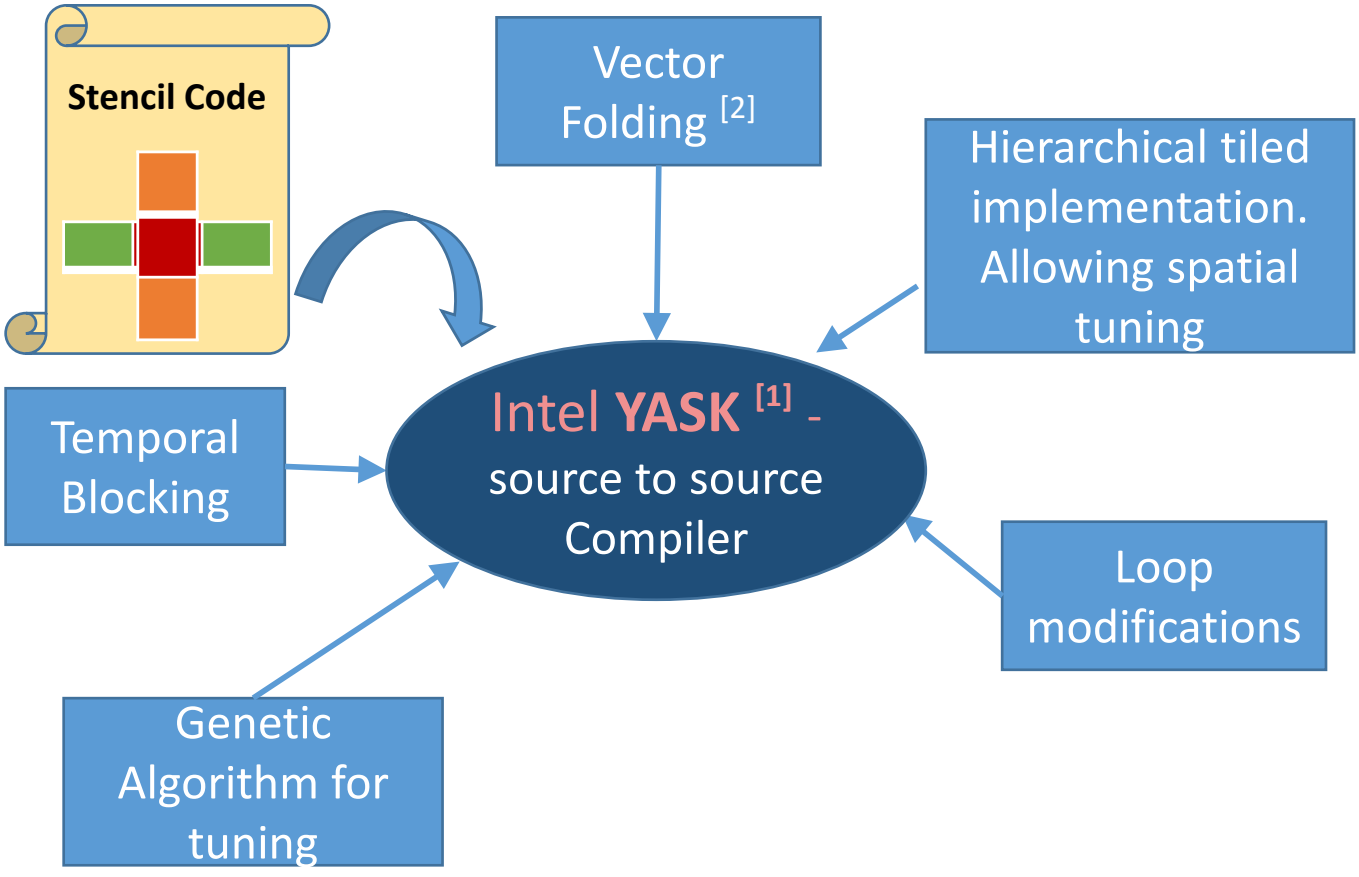
- Die Energiemessungen wurden mithilfe von RAPL durchgeführt
- U-Form für alle Systeme erkennbar für den Energieverbrauch
- Das Desktop-System (Skylake) verbraucht die wenigste Energie

Outline

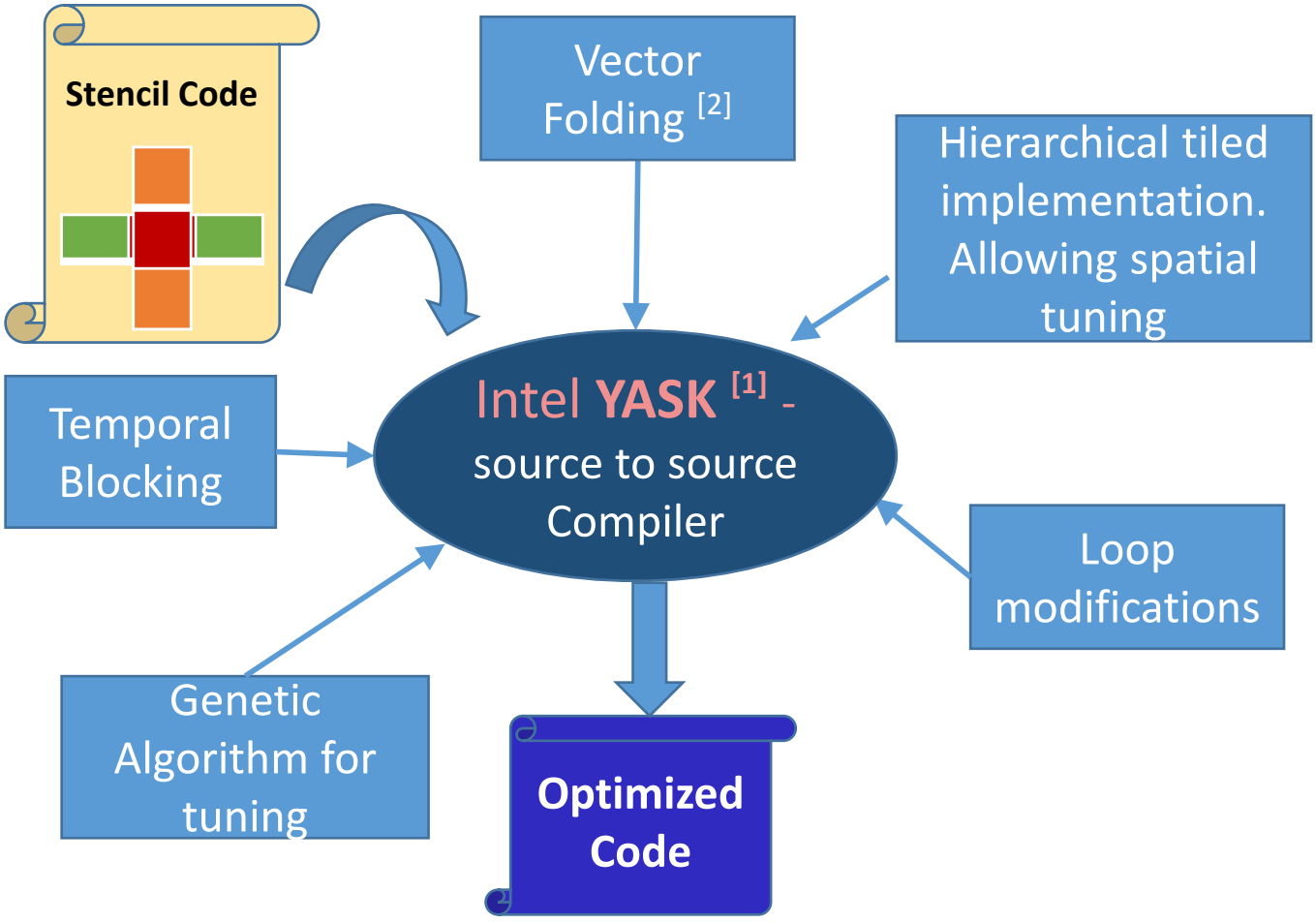
[1] . https://github.com/01org/yask



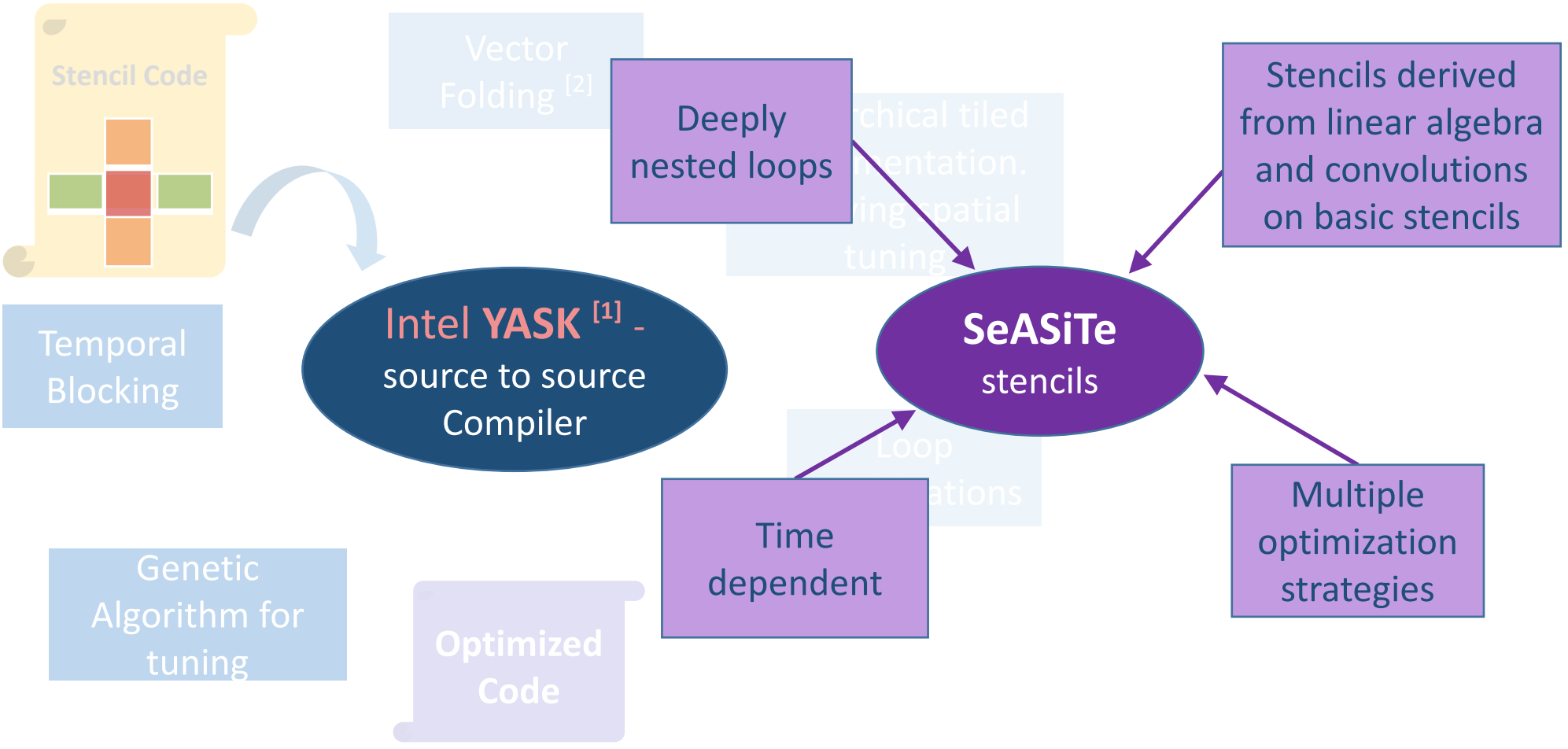
[1] . https://github.com/01org/yask



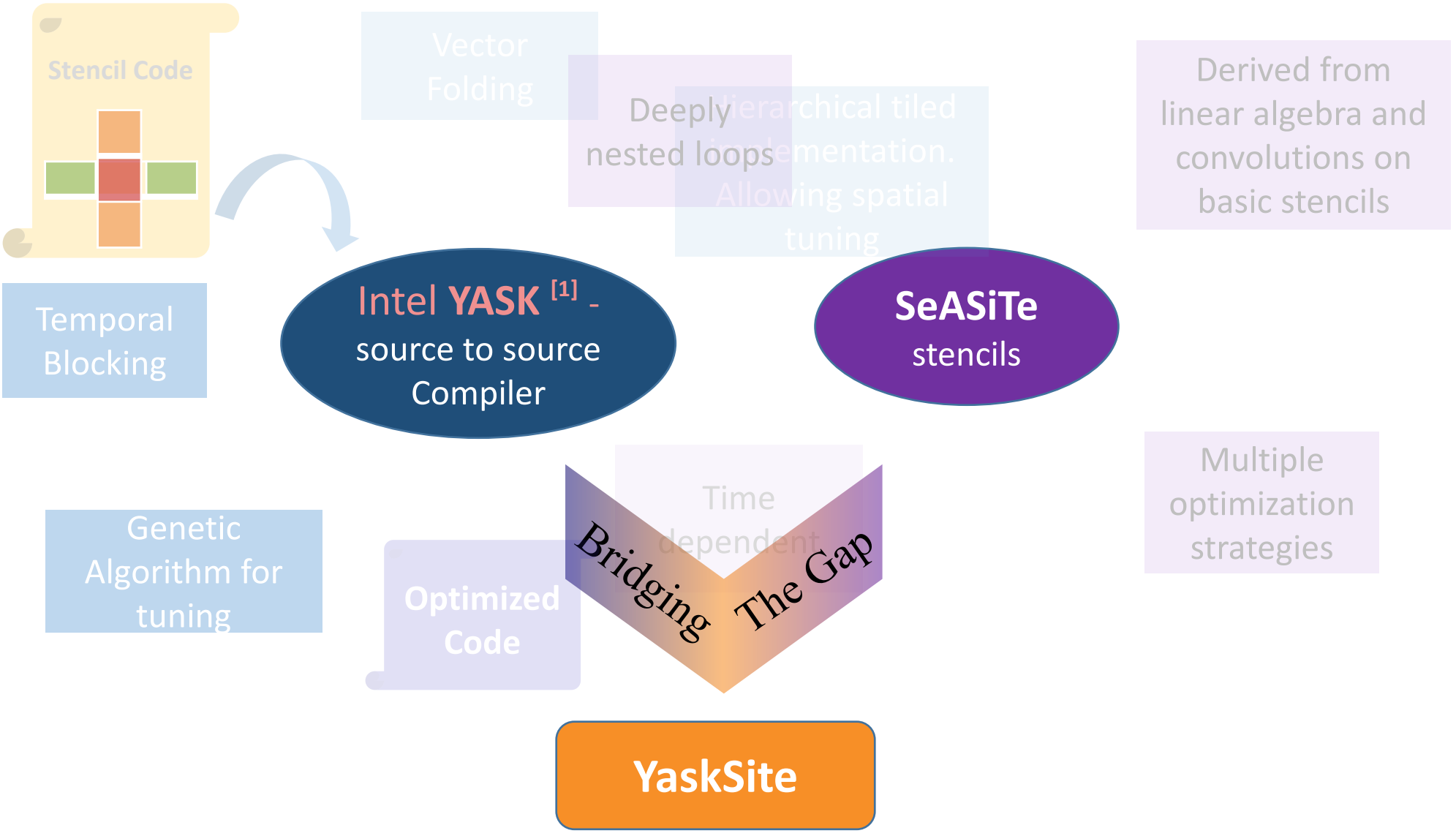
[1] . https://github.com/01org/yask
[2]. Yount, Chuck. (2015). Vector Folding: Improving Stencil Performance via Multi-dimensional SIMD-vector Representation. . 10.1109/HPCC-CSS-ICESS.2015.27.



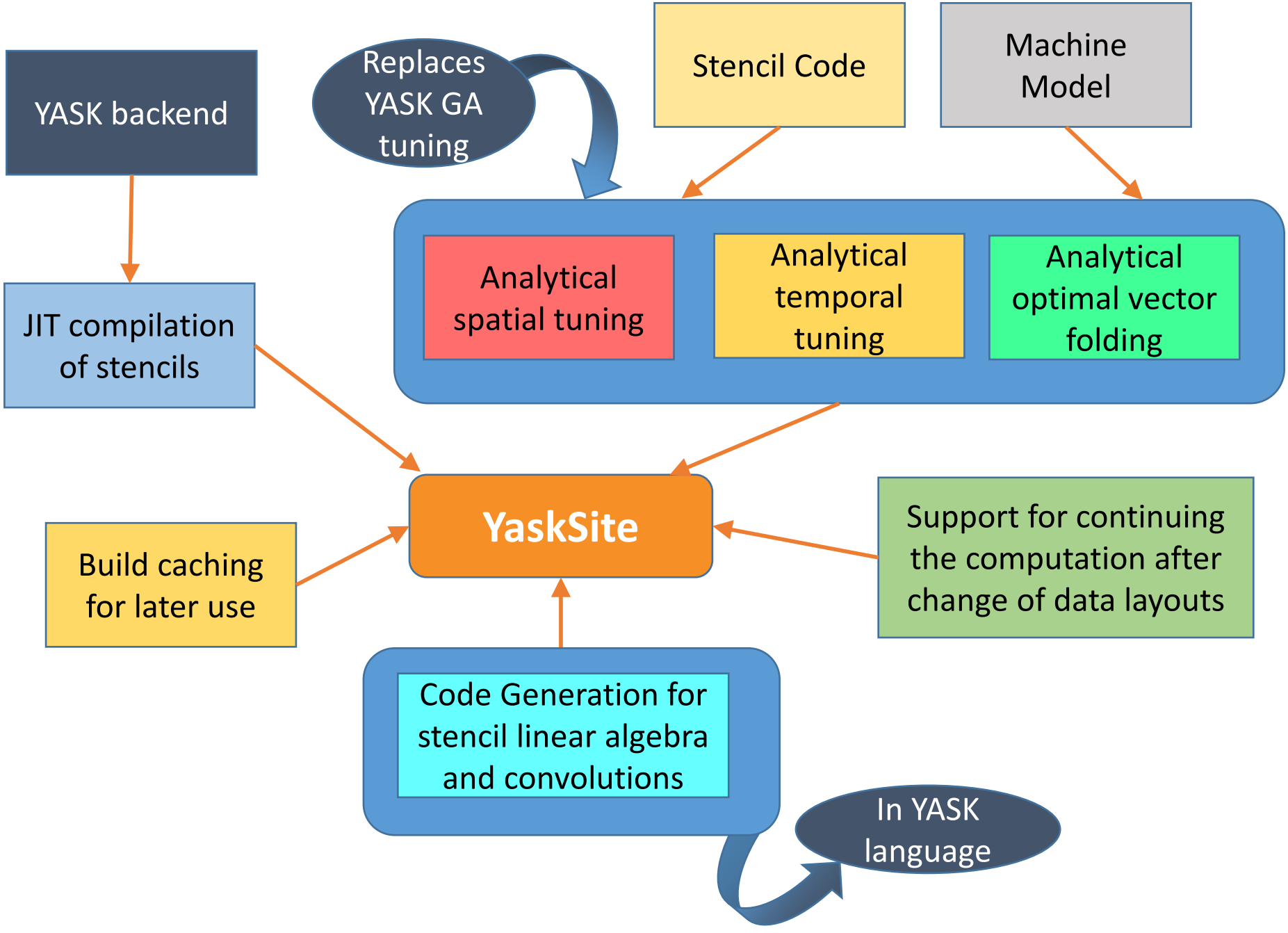
[1] . https://github.com/01org/yask
[2]. Yount, Chuck. (2015). Vector Folding: Improving Stencil Performance via Multi-dimensional SIMD-vector Representation. . 10.1109/HPCC-CSS-ICESS.2015.27.

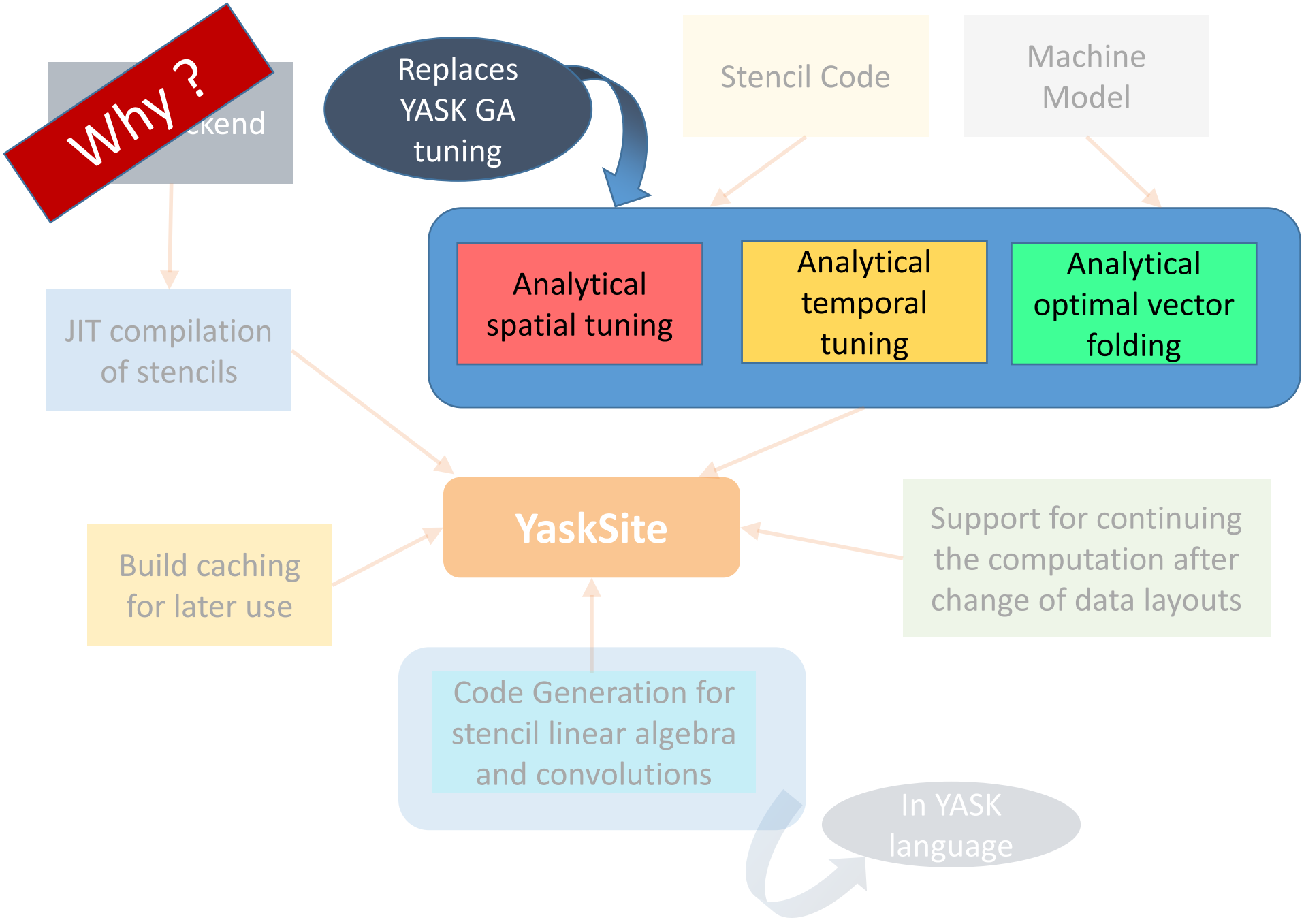


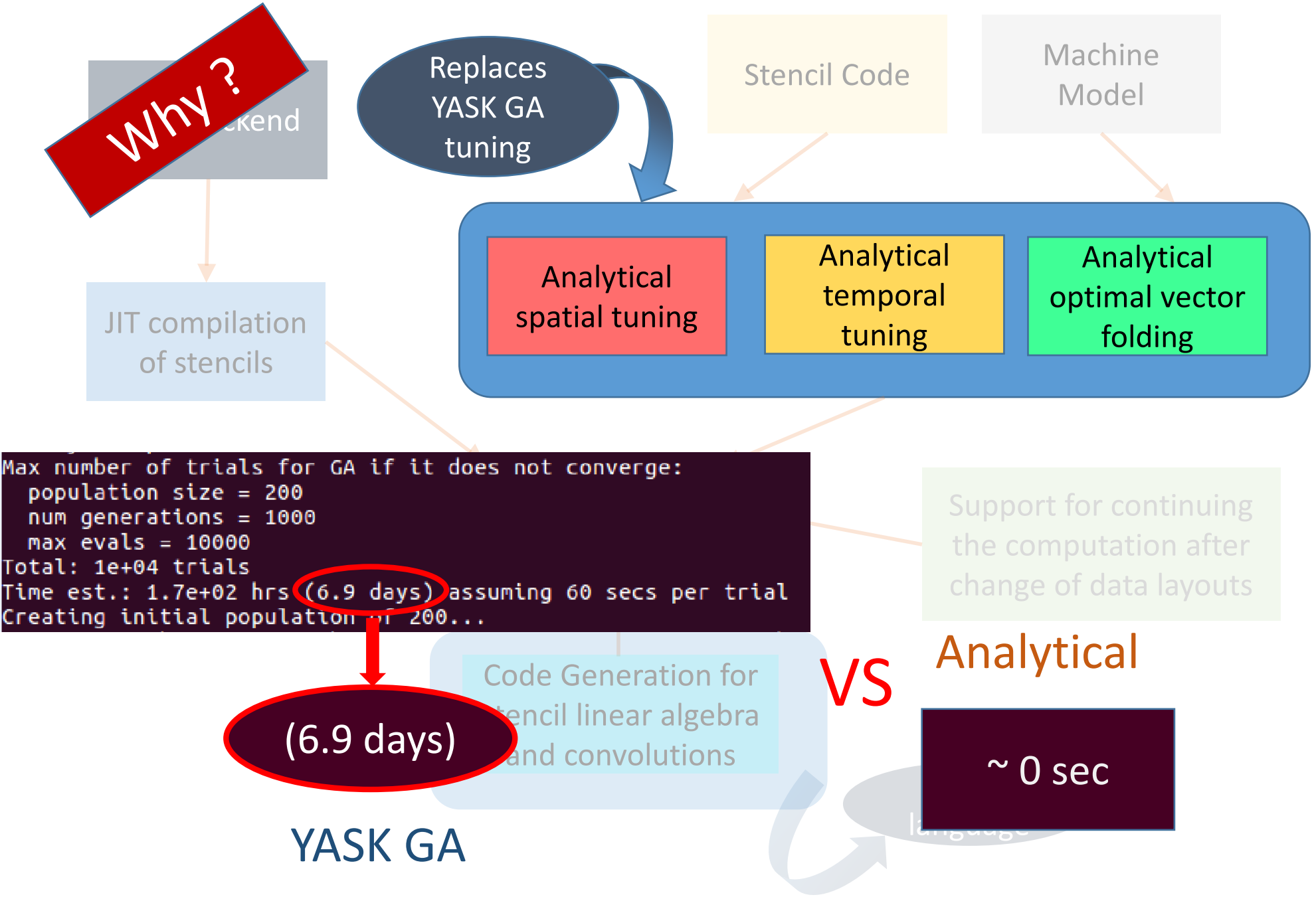
[1] . https://github.com/01org/yask
[2]. Yount, Chuck. (2015). Vector Folding: Improving Stencil Performance via Multi-dimensional SIMD-vector Representation. . 10.1109/HPCC-CSS-ICESS.2015.27.



[1] . https://github.com/01org/yask
[2]. Yount, Chuck. (2015). Vector Folding: Improving Stencil Performance via Multi-dimensional SIMD-vector Representation. . 10.1109/HPCC-CSS-ICESS.2015.27.


YASK backend
Replaces YASK GA tuning
Stencil Code
Machine Model
JIT compilation of stencils
Analytical spatial tuning
Analytical temporal tuning
Analytical optimal vector folding
YaskSite
Build caching for later use
Support for continuing the computation after change of data layouts
Code Generation for stencil linear algebra and convolutions
In YASK language


Why ?
Replaces YASK GA tuning
Stencil Code
Machine Model
Analytical spatial tuning
Analytical temporal tuning
Analytical optimal vector folding
JIT compilation of stencils
YaskSite
Build caching for later use
Support for continuing the computation after change of data layouts
Code Generation for stencil linear algebra and convolutions
In YASK language

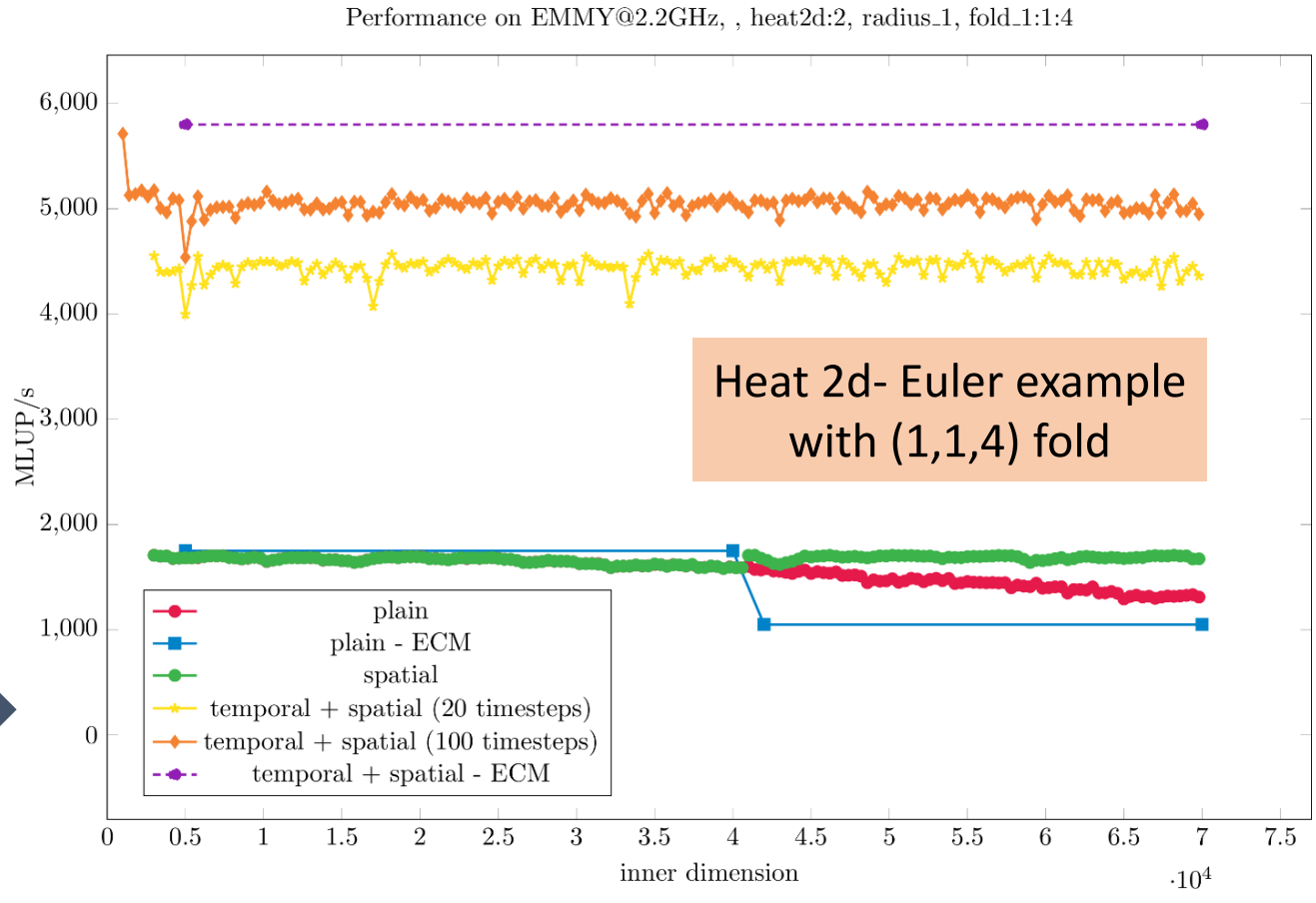

Why ?
Replaces YASK GA tuning
Stencil Code
Machine Model
JIT compilation of stencils
Analytical spatial tuning
Analytical temporal tuning
Analytical optimal vector folding
Max number of trials for GA if it does not converge:
population size = 200
num generations = 1000
max evals = 10000
Total: 1e+04 trials
Time est.: 1.7e+02 hrs (6.9 days) assuming 60 secs per trial
Creating initial population of 200...
(6.9 days)
YASK GA
Support for continuing the computation after change of data layouts
Code Generation for
and convolutions
VS
Analytical
~ 0 sec

Easy tuning

```
//spatial blocked
stencil_1->spatialTuner("L3", "L2");
stencil_1->run;

//temporal & spatial blocked
stencil_1->blockTuner("L3","L3","L2");
stencil_1->run;
```

Performance close to ECM model.




```
//2. RK(f23) method
CODIFY( "heat2d_rk", "heat2d",
        "GRID k1, k2;\n"
        "GRID_POINT k3;\n"
        "PARAM dt;\n"
        "k1   = STENCIL(INP);\n"
        "k2   = k1 + dt*1.0*STENCIL(k1);\n"
        "k3   = k1 + (0.25)*(k2-k1) + dt*0.25*STENCIL(k2);\n"
        "OUT = INP + dt*(0.5*k1 + 0.5*k2 + 0*k3);\n"
      );
```

Example RK method

Simple Code Generation

Outline

loop_adapt

Manipulation der Laufzeitumgebung und Algorithmus zur Laufzeit

- Ziel:
  Kombination statischer Analyse mit Laufzeitanpassungen zur Bestimmung optimalerer System- und Algorithmusparameter
- Idee:
  Schleifenmanager zum Messen des Laufzeitverhaltens und Anpassung von System- und Algorithmusparametern
- Ausblick:
  Statische Performance Analyse mit `kerncraft` dient als Eingabe für den Laufzeitmanager mit `LIKWID` und anderen Backends

loop_adapt

Manipulation der Laufzeitumgebung und Algorithmus zur Laufzeit

- C-Bibliothek integrierbar in bestehende Applikationen
- Registrierung von Schleifen und mehreren Strategien pro Schleife
- Strategien bestimmen Messverfahren, die Auswertung der Ergebnisse und die Anpassung von Parametern
- Strategiebeispiele:
  - Hohe Last im Speicher-Subsystem: Reduzierung der CPU Taktrate
  - Niedrige Last im Speicher-Subsystem: Reduzierung der Uncore Taktrate
  - Überschreiten der Cachegrößen: Reduzierung des Blockungsfaktors / Wechsel auf einen Algorithmus mit besserer Wiederverwendung der Daten.
- Applikationsspezifische Metriken wie MLUP/s oder *Partikel pro Sekunde* in Strategien verwendbar

loop_adapt

Manipulation der Laufzeitumgebung und Algorithmus zur Laufzeit

Wie es funktioniert:

1. Generierung des Performance Modells und der Suchräume (Offline)
2. Einlesen des Modells und der Suchräume beim Start der Applikation
3. Messen der Schleifendurchläufe mit `LIKWID`, Zeitmessung, ...
4. Auswerten der Ergebnisse und ggfs. Parameteranpassungen
5. Nach den Optimierungsversuchen läuft die Applikation mit den aktuell besten Einstellungen
6. Ausgabe der reduzierten Suchräume und besten Parameterwerte
7. Erneute Applikationsausführung mit gleichen Einstellungen: gehe zu 2
   Erneute Applikationsausführung mit anderen Einstellungen: gehe zu 1

Zusammenfassung

- Autotuning-Mechanismus wird unterteilt in **Offline-Phase und Online-Phase**.
- In der **Offline-Phase** werden verschiedene **Implementierungsvarianten** erzeugt und mit einem **Performancemodell** bewertet.
- In der **Online-Phase** werden die **ersten Zeitschritte** dazu verwendet, aus den vielversprechendsten Implementierungsvarianten die **schnellste Implementierungsvariante** zu ermitteln.
  Diese wird dann für die **folgenden Zeitschritte** verwendet.
- Durch **Monitoring** können Änderungen des Ausführungsverhaltens **dynamisch erfasst** werden.
- Unterschiedliche zeitschrittbasierte Simulationsverfahren erfordern eine **unterschiedliche Aufteilung zwischen Offline-Phase und Online-Phase**.