W. Karl, J. Keller (Eds.): PARS 2017
Proc. 27th PARS-Workshop

Cache-Partitionierung im Kontext von Co-Scheduling

Josef Weidendorfer[1], Carsten Trinitis[2], Sebastian Rückerl[3] und Michael Klemm[4]

Abstract: Neuere Mehrkern-Architekturen, die allen Rechenkernen einen gemeinsam nutzbaren Cache zur Verfügung stellen, besitzen die Fähigkeit, diesen Cache dynamisch zwischen den Kernen aufzuteilen. Die Partitionierbarkeit ist dafür gedacht, sogenannten Cloud-Anbietern zu erlauben, einzelne Kerne an Kunden zu vermieten, ohne dass deren Rechenlasten sich gegenseitig beeinflussen oder Seitenkanäle zum Abgreifen von Daten entstehen. Cache-Partitionierung lässt sich aber auch gewinnbringend dafür einsetzen, mehrere Anwendungen aus dem Hochleistungsrechnen so auf Mehrkern-Architekturen ablaufen zu lassen, dass sie sich gegenseitig auf der gemeinsam nutzbaren Cache-Ebene nicht stören und dynamisch eine jeweils passende Cache-Größe zur eigenen Verfügung haben.
In diesem Beitrag werden erste Ergebnisse zur Cache-Partitionierung mittels Cache Allocation Technology CAT) und deren Auswirkungen auf Co-Scheduling-Strategien im Hochleistungsrechnen vorgestellt.

Keywords: Co-Scheduling, Cache, Partitionierung, High Performance Computing

1 Einleitung

Die meisten derzeit im Hochleistungsrechnen verwendeten Mehrkernprozessorsysteme verfügen über Prozessorarchitekturen, bei denen der gemeinsame Last-Level Cache (LLC) vollumfänglich allen auf dem Prozessor laufenden Anwendungen zur Verfügung steht. Neuere x86-basierte Server-Prozessorarchitekturen - und damit die Standard-Prozessorarchitekturen in künftigen Exascale-Systemen - sind jedoch in der Lage, diese bisher ausschließlich gemeinsam nutzbare Ressource zu partitionieren. Dadurch ist es möglich, bei Bedarf Programme besser voneinander zu isolieren. Dies kann insbesondere dann von Vorteil sein, wenn auf den Knoten sogenanntes Co-Scheduling (siehe auch Kapitel 2) zur Anwendung kommt [BWT15].

Dies kann insbesondere für Rechenzentren von Interesse sein, wenn es darum geht, den Durchsatz der Anwenderprogramme zu erhöhen. Bereits heute erlauben bestimmte Intel-Serverprozessoren seit der Haswell EP-Baureihe, den L3-Cache zwischen mehreren auf dem Mehrkernprozessor laufenden Anwendungen aufzuteilen. Dadurch kann eine Beeinflussung und damit gegenseitige Störung von mittels Co-Scheduling auf einem Knoten laufenden Programmen verringert werden. Erste Untersuchungen hierzu werden in diesem Beitrag präsentiert. Dabei liegt der Fokus insbesondere auf der Analyse der zu beobachtenden Effekte, was mit Hilfe zweier gängiger Benchmarks auf einem zum Zeitpunkt der Messungen verfügbaren Zweisockelsystem untersucht wurde.

[1] TU München, Fakultät für Informatik, Boltzmannstr. 3, 85748 Garching, Josef.Weidendorfer@tum.de
[2] TU München, Fakultät für Informatik, Boltzmannstr. 3, 85748 Garching, Carsten.Trinitis@tum.de
[3] TU München, Lehrstuhl für Raumfahrttechnik, 85748 Garching, Sebastian.Rueckerl@tum.de
[4] Intel Deutschland GmbH, Dornacher Str. 1, 85622 Feldkirchen, michael.klemm@intel.com

Der Beitrag ist wie folgt aufgebaut: Kapitel 2 gibt einen Überblick über verwandte Arbeiten, Kapitel 3 erläutert in groben Zügen, wie Cache-Partitionierung funktioniert. Kapitel 4 umschreibt, welche Anwendungen als geeignete Benchmarks für die Untersuchungen ausgesucht wurden und beschreibt kurz das Werkzeug `distgen`. Kapitel 5 präsentiert die durchgeführten Messungen, die in Kapitel 6 diskutiert werden. Kapitel 7 fasst die Ergebnisse zusammen. Im Folgenden wird für die oben beschriebene neu zu untersuchende Funktionalität der Cache-Partitionierung der Begriff "Cache Allocation Technology", kurz CAT, benutzt, welcher auch von einschlägigen Herstellern verwendet wird.

2 Verwandte Arbeiten

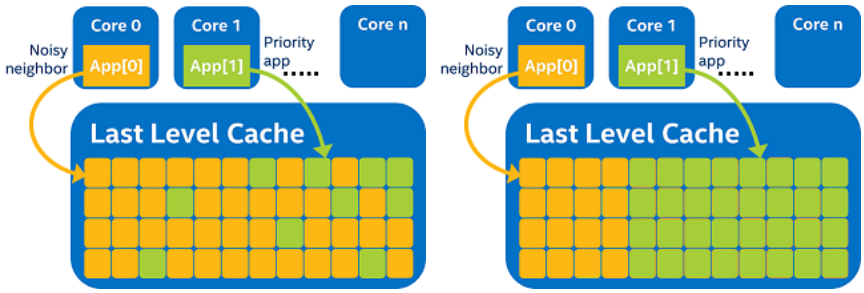
Eine wichtige Voraussetzung zum effizienten Einsatz der im folgenden Kapitel näher erläuterten Cache Allocation Technology im Bereich des Hochleistungsrechnens stellt das sogenannte *Co-Scheduling* dar. Hierbei werden mehrere (in der Regel jedoch zwei) Anwendungen auf einem Rechenkonten gleichzeitig ausgeführt. Wenn diese Anwendungen unterschiedliche Anforderungen an die Ressourcen stellen (z.B. speichergebunden vs. rechengebunden), lässt sich mit Hilfe von Co-Scheduling bei geschickter Ressourcenvergabe eine Durchsatzsteigerung gegenüber herkömmlichen Verfahren erzielen. Erste Untersuchungen mit Co-Scheduling werden in [BM10, Ha14] präsentiert. Forschungsarbeiten zur entsprechenden Charakterisierung von Anwendungen sind beispielsweise in [Ha16] zu finden, passende Schedulingverfahren werden in [Su16, BL16, dBL17, PBL17] behandelt. In [CME16] wird die Integration von Co-Scheduling auf Systemebene untersucht. Eine Verbesserung von Co-Scheduling mittels CAT-Technologie wird in [Pa17] erläutert.

3 Cache Allocation Technology (CAT)

Intels Cache Allocation Technology (CAT) erlaubt es dem Anwender, softwareseitig zu spezifizieren, wo sich die Daten im Last-Level Cache befinden sollen. Dadurch lassen sich Anwendungen mit unterschiedlichen Ressourcenanforderungen im LLC voneinander isolieren, und es können ihnen unterschiedliche LLC-Cache-Größen zugewiesen werden.

Die ursprüngliche Motivation für eine Cache-Partitionierung kommt aus dem Bereich des Cloud-Computing: In heutigen Rechenzentren, in denen virtuelle Maschinen mit unterschiedlichen Anwendungen im Cloud-Betrieb laufen, ist es wichtig, optimale Leistung und Priorisierung zu gewährleisten. Zahlreiche gemeinsam genutzte Ressourcen existieren dabei innerhalb eines Rechenzentrums (z.B. die Netzwerkverbindung), und innerhalb eines Knotens basierend auf aktuellen Mehrkernprozessoren kann dies beispielsweise der LLC sein.

Bestimmte Anwendungen können dabei von einem größeren LLC profitieren, wogegen andere wiederum fast ausschließlich auf dem Hauptspeicher rechnen und dadurch durch eine größere LLC-Partition ohnehin keine Leistungsverbesserung zeigen. Abb. 1a (ohne Cache-Partitionierung) zeigt auf Kern 0 hellbraun markiert eine Anwendung, die exzessi-



(a) Ohne Cache-Partitionierung (b) Mit Cache-Partitionierung

Abb. 1: Last Level Cache (LLC) Nutzung mehrerer zeitgleich laufender Anwendungen

ven LLC-Bedarf aufweist und dadurch die grün gekennzeichnete Anwendung auf Kern 1 ausbremst[3].

Wird nun der LLC so aufgeteilt, dass jede der beiden Anwendungen eine exklusive Partition zugewiesen bekommt, lassen sich die oben beschriebenen Verdrängungseffekte vermeiden (Abb. 1b). Welche Arten von Anwendungen welche Partitionsgröße zugewiesen bekommen sollten, wird im Folgenden untersucht.

4 Klassifizierung von geeigneten Programmen

Im Falle von Co-Scheduling (Ablauf mehrerer Programme auf einem Knoten) ist darauf zu achten, dass sich diese möglichst wenig gegenseitig behindern. Um weiterhin von der Cache-Partitionierung zu profitieren, ist es vorteilhaft, wenn es sich um Anwendungen handelt, bei denen die eine von großem Cache-Speicher profitiert, und die andere nicht viel Cache-Speicher benötigt bzw. diesen nicht sinnvoll nutzen kann, etwa wenn die Daten deutlich größer als der Cache-Speicher sind. Obwohl in letzterem Fall der Cache nicht vorteilhaft ist, wird er dennoch belegt und reduziert die Vorteile für andere auf dem Knoten laufende Programme. Eine Cache-Partitionierung, die diese unnötige Belegung des Cache-Speichers verhindert, ist deshalb besonders von Vorteil.

Zur ersten Untersuchung und Klassifizierung wurden hierzu die sogenannten STREAM-Benchmarks [Mc95][4] herangezogen. Die STREAM-Benchmarks stellen den De-Facto-Standard zur Messung von Speicherbandbreite eines Systems dar und sind daher für eine entsprechende Klassifizierung gut geeignet. Mit den STREAM-Benchmarks lässt sich der Speicherdurchsatz in MByte/s für vier gängige Standardoperationen ermitteln:

COPY (A=C), SCALE (A=s*C), ADD (A=B+C), und TRIAD (A=B+s*C)

Dabei sind A, B und C Vektoren gleicher Länge, die in den Untersuchungen variiert wurde. Die Elemente der Vektoren sind Fließkommawerte doppelter Genauigkeit. Von den

[3] https://software.intel.com/en-us/articles/introduction-to-cache-allocation-technology

[4] http://www.cs.virginia.edu/stream

STREAM-Benchmarks wurden jeweils zwei gleiche Operationen (also COPY-COPY etc.) auf einem Knoten untersucht.

Weiterhin wurden neben den STREAM-Benchmarks auch die NAS Parallel Benchmarks (NPB) [Ba91] analysiert. Dabei kam für jeden NPB als Gegenläufer das eigens dazu entwickelte Werkzeug `distgen` (siehe Kapitel 5) zum Einsatz. Bei den NPBs handelt es sich um eine Reihe von Benchmarks, die darauf abzielen, die Leistungsfähigkeit hochparalleler Supercomputer zu ermitteln. Da diese die Speicherhierarchie auf unterschiedlichste Art und Weise nutzen, sind sie gut geeignet, die Effekte der Cache-Partitionierung genauer zu analysieren. Zum Einsatz kamen die folgenden auf der NPB-Webseite beschriebenen Pakete:

- *IS - Integer Sort, random memory access*
- *EP - Embarrassingly Parallel*
- *CG - Conjugate Gradient, irregular memory access and communication*
- *MG - Multi-Grid on a sequence of meshes, long- and short-distance communication, memory intensive*
- *FT - discrete 3D fast Fourier Transform, all-to-all communication*
- *BT - Block Tri-diagonal solver*
- *SP - Scalar Penta-diagonal solver*
- *LU - Lower-Upper Gauss-Seidel solver*
- *DC - Data Cube*
- *UA - Unstructured Adaptive mesh, dynamic and irregular memory access*

Die NBP-Benchmarks wurden unter anderem deshalb ausgewählt, weil deren Verhalten beim Co-Scheduling (ohne Verwendung von Cache-Partitionierung) schon in vorausgehenden Untersuchungen im Projekt FAST[5] analysiert wurde [BWT16]. Dabei ist deutlich zu erkennen, dass sich die Benchmarks sehr unterschiedlich beeinflussen, wenn sie gemeinsam auf einem Teil desselben Systems ausgeführt werden.

5 Das Werkzeug `distgen`

Das an der TU München entwickelte Werkzeug `distgen` ist darauf ausgelegt, bestimmte Ressourcen eines Rechensystems (z.B. Speicherbus und Cache) gezielt zu belasten. Auf eine konfigurierte Speichergröße wird so oft wie möglich wiederholt zugegriffen. Da jeder moderne Prozessor eine mehrstufige Cache-Hierarchie besitzt, werden bei den wiederholten Zugriffen nur die entsprechend großen Caches in dieser Hierarchie genutzt. Aktuelle Intel-Prozessoren besitzen von den Kernen dediziert genutzte L1- und L2-Caches mit

[5] `http://fast-project.de`

32 KB und 256 KB Größe, sowie einen von allen Kernen im Prozessor genutzten LLC (L3-Cache) mit mehreren MB Größe. Solange also `distgen` mit Speichergröße kleiner 256 KB konfiguriert ist (und damit nur auf seine privaten Cache-Ebenen zugreift), sollte dies keine Auswirkung auf die Leistung anderer Kerne haben. Bei größerer Speicherkonfiguration hingegen wird zunächst der LLC verwendet und danach auf den Hauptspeicher zugegriffen. Diese beiden Komponenten der Speicherhierarchie werden also gemeinsam genutzt, womit `distgen` prinzipiell die auf anderen Kernen laufenden Programme beeinflusst. Dies hängt jedoch auch von der Nutzung der Speicherhierarchie der anderen Programme ab.

Mit `distgen` kann also ein bestimmtes Verhalten von Anwendungen simuliert werden. `distgen` wurde im Rahmen des Projektes FAST entwickelt und dort als Co-Scheduling-Partner mit größtmöglicher Beeinflussung der anderen Anwendung benutzt, indem es Komponenten der Speicherhierarchie hochgradig belastet. Dadurch können Anwendungen sowohl bezüglich ihrer eigenen Empfindlichkeit als auch hinsichtlich ihres Einflusscharakters gegenüber anderen Anwendungen detailliert analysiert werden [WB16, BW17].

6 Messergebnisse

Die Messungen erfolgten auf einem Intel(R) Xeon(R)-System mit 2 CPU E5-2699 v4, Taktfrequenz 2.20GHz und 128 GB DDR 2400-Hauptspeicher in 16 GB-Modulen.

Bei Beginn der Untersuchungen wurde zur Steuerung der neuartigen Funktionalität auf entsprechende Open-Source-Software von Intel zurückgegriffen[6], da eine Unterstützung in den Linux-Quellen noch nicht verfügbar war. Im weiteren Verlauf der Untersuchungen war eine erste Version einer geplanten, offiziell in Linux integrierten Unterstützung von CAT verfügbar[7].

Auf einem Knoten wurden jeweils zwei Anwendungen (Haupt- und Gegenläufer) gegeneinander gestartet, um so dabei auftretende Effekte genauer untersuchen zu können.

STREAM-Benchmarks Bei den STREAM-Benchmarks handelt es sich um jeweils zwei Funktionen mit Zugriff auf zwei Operanden (COPY/SCALE: ein Ein- und ein Ausgabeoperand) bzw. drei Operanden (ADD/TRIAD: zwei Ein- und ein Ausgabeoperand). Da das Verhalten bei COPY und SCALE bzw. bei ADD und TRIAD aufgrund des identischen Zugriffsmusters mehr oder weniger gleich ist, werden im Folgenden exemplarisch jeweils nur COPY für zwei bzw. TRIAD für drei Operanden diskutiert. In diesem Fall ist zu beachten, dass Haupt- und Gegenläufer gleich sind, jedoch die Ein-/Ausgabedatengrößen variiert werden.

Die Abbildungen 2 bis 5 zeigen dabei die Laufzeit für den COPY-Benchmark (Hauptläufer) gegen sich selbst (Gegenläufer). Die verschiedenen Kurven beziehen sich auf die

[6] "pqos", siehe `https://github.com/01org/intel-cmt-cat`
[7] `https://lkml.org/lkml/2016/10/30/308`

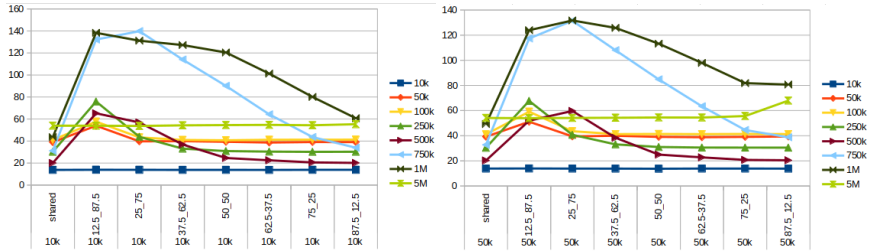


Abb. 2: COPY mit 10 KB (links) / 50 KB (rechts) Ein-/Ausgabedaten

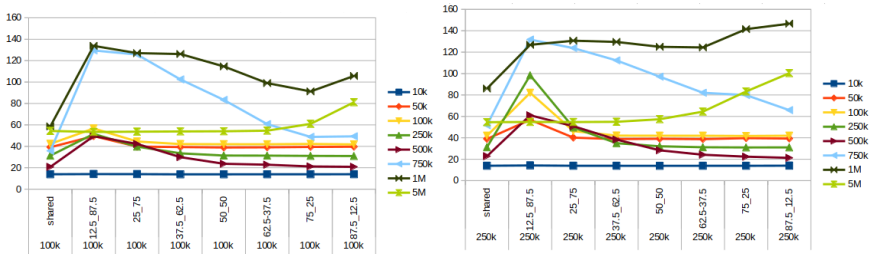


Abb. 3: COPY mit 100 KB (links) / 250 KB (rechts) Ein-/Ausgabedaten

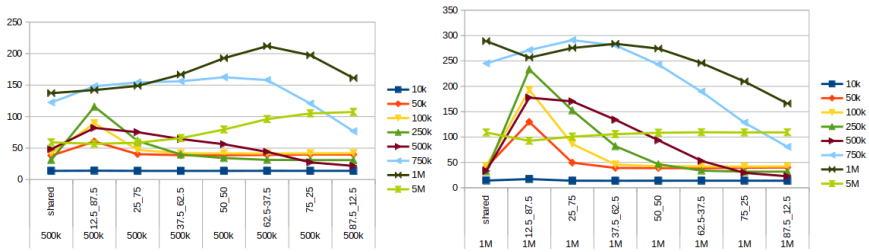


Abb. 4: COPY mit 500 KB (links) / 1 MB (rechts) Ein-/Ausgabedaten

Ein-/Ausgabedatengrößen des Hauptläufers; die verschiedenen Diagramme beziehen sich auf die Ein-/Ausgabedatengrößen des Gegenläufers. Die x-Achsenbeschriftung bedeutet dabei die Cache-Partitionierung Hauptläufer vs. Gegenläufer, also z.B. 12.5 - 87.5 entspricht einer Zuteilung von einem Achtel des L3-Cache für den Hauptläufer und sieben Achtel für den Gegenläufer. Der erste Wert “shared” ganz links bedeutet wie oben, dass keine Cache-Partitionierung eingestellt ist. Die y-Achsenbeschriftung zeigt die Laufzeit des Hauptläufers an.

Die Abbildungen 6 bis 9 zeigen die für TRIAD ähnlich durchgeführten Messungen. Wie oben erwähnt ist dabei TRIAD ein Beispiel für drei Ein-/Ausgabeparameter, was einen um 50 % erhöhten Speicherbedarf nach sich zieht.

Wiederum kann festgestellt werden, dass die Laufzeit zunächst im Vergleich zu „shared“ ansteigt. Sobald jedoch der Hauptläufer von der hinreichend großen L3-Cache-Partition profitiert, sinkt die Gesamtlaufzeit wieder. Die Bremseffekte durch den Gegenläufer ent-

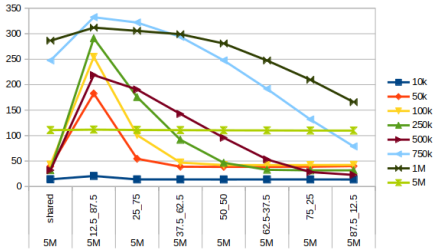


Abb. 5: COPY mit 5 MB Ein-/Ausgabedaten

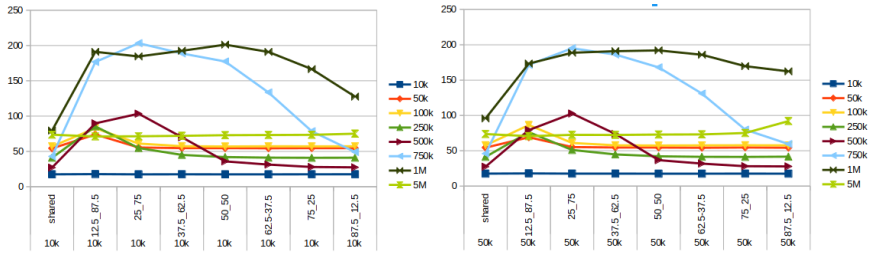


Abb. 6: TRIAD mit 10 KB (links) / 50 KB (rechts) Ein-/Ausgabedaten

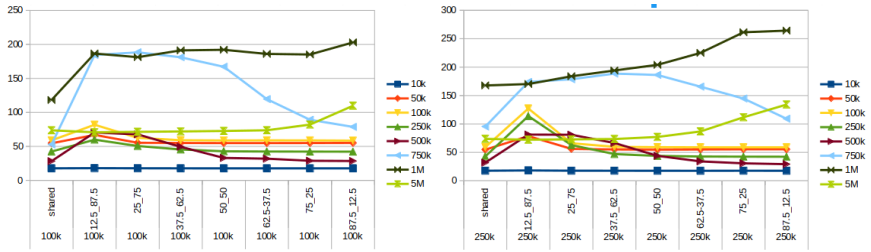


Abb. 7: TRIAD mit 100 KB (links) / 250 KB (rechts) Ein-/Ausgabedaten

stehen wiederum dadurch, dass dieser nicht in seine L3-Cache-Partition passt und somit auf den Hauptspeicher zugreifen muss.

Je nach Konfiguration (Größe der Ein-/Ausgabedaten von Haupt- und Gegenläufer sowie prozentuale Partitionierung des L3-Cache) verschiebt sich das Laufzeitmaximum nach rechts oder links. In den Fällen, in denen die Laufzeit unter die ganz links unter ”shared” eingetragene Laufzeit sinkt, lässt sich eine Leistungssteigerung durch Cache-Partitionierung erzielen.

NAS Parallel Benchmarks (NPBs) Als nächstes wurden die ausgewählten NPB-Benchmarks (Hauptläufer) jeweils gegen `distgen` (Gegenläufer) bei unterschiedlichen Cache-Partitionierungen gemessen. In der Legende der folgenden Abbildungen ist neben dem Benchmark-Namen jeweils die verwendete Klasse (C oder D) der Eingabegröße angegeben. Weiterhin wurde der von `distgen` benötigte Speicher variiert: 0,2 MB, 0,8 MB,

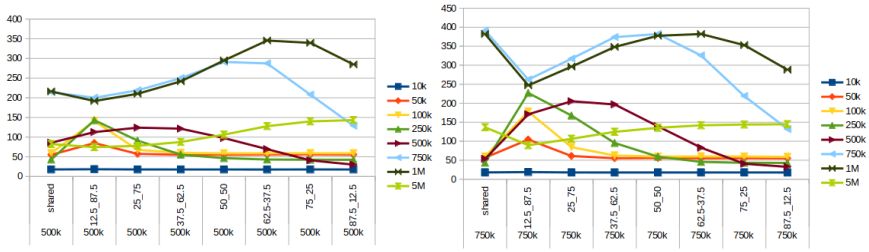


Abb. 8: TRIAD mit 500 KB (links) / 750 KB (rechts) Ein-/Ausgabedaten

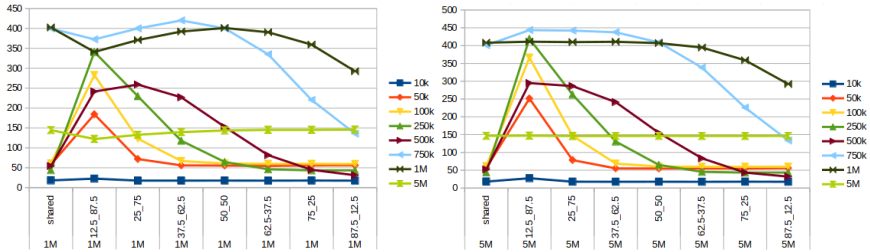


Abb. 9: TRIAD mit 1 MB (links) / 5 MB (rechts) Ein-/Ausgabedaten

1,6 MB, 3,2 MB, 6,4 MB, 12,8 MB sowie 500 MB. Diese Variation erlaubt es, das Verhalten der NPB-Benchmarks bei unterschiedlichem Zugriffsverhalten von Gegenläufern zu analysieren.

Wie in Abschnitt 5 beschrieben, greift ein `distgen` mit 0,2 MB konfiguriertem Speicherbedarf nur auf den L1/L2-Cache des Prozessorkerns zu, auf dem er selbst läuft. Bei 500 MB hingegen wird ein Gegenläufer simuliert, der selbst ständig auf den Hauptspeicher zugreift und die Speicherbandbreite nach Möglichkeit für sich selbst ausnutzt. Die Abbildungen 10 bis 13 zeigen für die untersuchten NPBs jeweils die Verlangsamungsfaktoren gegenüber `distgen` mit verschiedenem Speicherbedarf für verschiedene Cache-Partitionierungen. Die x-Achsenbeschriftung bedeutet dabei die Cache-Partitionierung des Benchmarks gegenüber `distgen`, also z.B. 12.5 - 87.5 entspricht einer Zuteilung von einem Achtel des L3-Cache für den jeweiligen Benchmark und sieben Achtel für `distgen`. Der erste Wert „shared“ bedeutet, dass keine Cache-Partitionierung eingestellt ist. Die y-Achsenbeschriftung zeigt die Laufzeit relativ zum Lauf nur des Benchmarks an (ohne `distgen`).

Ein Benchmark profitiert dann von Partitionierbarkeit des Caches, wenn es eine Partitionierung gibt, in der er schneller ist als im Fall der deaktivierten Partitionierung („shared“ in den Abbildungen). Dies hängt offensichtlich vom Benchmark selbst sowie dem Verhalten des Gegenläufers ab. Allgemein fällt in nahezu allen Diagrammen auf, dass im Vergleich zum Lauf ohne Cache-Partitionierung die Laufzeit zunächst ansteigt, dann mit zunehmendem Cache-Anteil für den Benchmark absinkt, und dann wieder ansteigt.

Dieses Verhalten lässt sich damit erklären, dass der steigende Anteil am L3-Cache sich zunächst positiv auswirkt, ab einer gewissen Schwelle aber `distgen` aufgrund des geringeren ihm zugewiesenen Cache-Anteils so viele Cache-Fehler verursacht, dass die dadurch

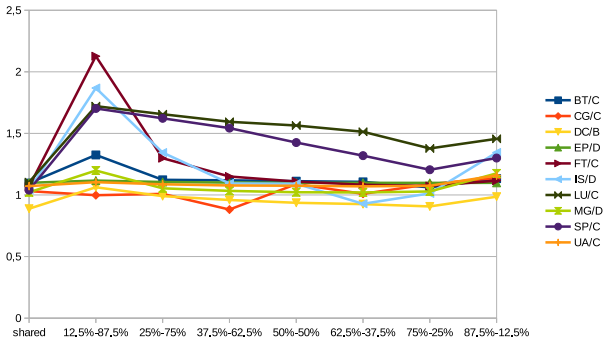


Abb. 10: `distgen` mit 0,2 MB Speicherbedarf

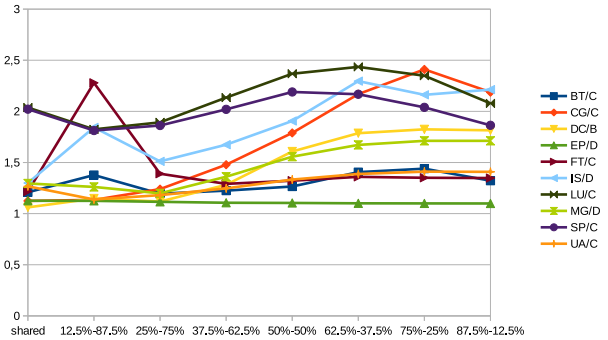


Abb. 11: `distgen` mit 3,2 MB Speicherbedarf

entstehenden Hauptspeicherzugriffe einen Benchmark, der selbst Hauptspeicherzugriffe benötigt, verstärkt beeinflusst, womit die Laufzeit wieder ansteigt. Dies passiert umso eher, je mehr Speicher für `distgen` konfiguriert wird. Insgesamt ist aus den Messungen sehr gut zu erkennen, welcher Benchmark den L3-Cache oder den Hauptspeicher gar nicht benötigt und sich so kaum beeinflussen lässt (z.B. EP/D), sich von Hauptspeicherzugriffen eines Gegenläufers stark beeinflussen lässt (z.B. CG/C, siehe Abbildung 11), oder stark von mehr L3-Cache profitiert (z.B. SP/C, siehe Abbildungen 12 und 13).

7 Diskussion der Ergebnisse

Wie durch die Messungen im vorigen Abschnitt deutlich geworden ist, bringt Cache-Partitionierung in bestimmten Szenarien Vorteile, aber bei weitem nicht in allen Fällen. Dies ist jedoch auch nicht zu erwarten, da man die Größe des für eine Anwendung zur Verfügung stehenden L3-Cache beschneidet. Dies gilt insbesondere, wenn CAT zur Verbesserung von Co-Scheduling genutzt werden soll. Durch CAT kann man dort eine gegenseitige Beeinflussung reduzieren, solange diese sich auf den (gemeinsamen) L3-Cache beschränkt.

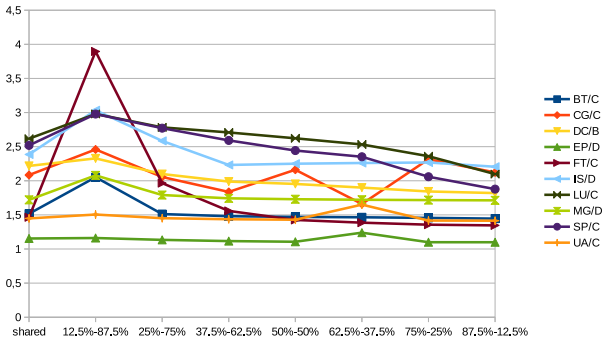


Abb. 12: `distgen` 12,8 MB Speicherbedarf

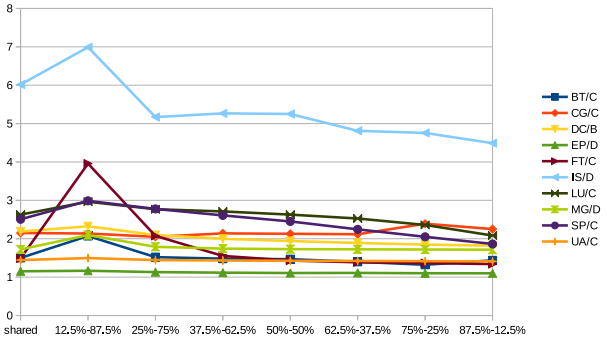


Abb. 13: `distgen` mit 500 MB Speicherbedarf

Das Problem von negativen Effekten auf die Leistung der Anwendung, die durch Hauptspeichernutzung entstehen, kann mit CAT alleine nicht gelöst werden. Wie oben detailliert erklärt, verändert Cache-Partitionierung jedoch das Hauptspeicherverhalten von Anwendungen (Programme, deren Daten ursprünglich in den L3 passen, müssen bei kleinerer Partition auf den Hauptspeicher ausweichen). Dies erfordert Bedacht bei der Verwendung von Cache-Partitionierung. Nur in gewissen Fällen ist ein Vorteil zu beobachten bzw. kann vorhergesagt werden, wenn das Nutzungsverhalten einer Anwendung in Bezug auf die Speicherhierarchie bekannt ist.

Im Wesentlichen ist dies dann der Fall, wenn eine Anwendung unnötigerweise die Daten der anderen Anwendung aus dem L3 verdrängt. Unnötig bedeutet hierbei, dass die Anwendung selbst vom L3-Cache nicht profitiert, etwa weil ihre Daten ohnehin deutlich größer sind und somit Hauptspeicherzugriffe erfordern. Hier kann CAT helfen, indem die Anwendung, die nicht von L3 profitiert, nur eine kleine Partition zugewiesen bekommt. Die andere Anwendung hat dadurch zwei Vorteile: Zum einen werden ihre Daten nicht verdrängt, zum anderen kann ihr eine große Partition dediziert zugewiesen werden. Im letzten Abschnitt konnte dieser Fall beobachtet werden, wenn sich die Laufzeit eines Benchmarks durch Partitionierung verbessert hat.

Nutung L3	Nein		Ja	
Nutzung Hauptspeicher	Nein	Ja	Nein	Ja
Beeinflusst (Co-Scheduling)	Nein	Ja, wenn andere L3 oder Hauptspeicher nutzt	Ja, wenn andere L3 oder Hauptspeicher	Ja, wenn andere Hauptspeicher nutzt
CAT vorteilhaft	Nein	Ja	Ja	Nein

Tab. 1: Nutzbarkeit von CAT

8 Zusammenfassung

Insgesamt ist zu beobachten, dass CAT nicht in jedem Fall bei Co-Scheduling zweier Anwendungen zwingend erforderlich ist; wie aber durch die Messungen an den NBP-Benchmarks gezeigt werden konnte, existieren durchaus Fälle, bei denen CAT eine deutliche Verbesserung der Leistung im Co-Scheduling-Fall erreichen kann. Diese hängt im wesentlichen davon ab, ob und wie gemeinsam nutzbare Ressourcen der Speicherhierarchie (L3-Cache, Hauptspeicherbandbreite) von Anwendungen tatsächlich genutzt werden. Tabelle 1 veranschaulicht diesen Zusammenhang.

In weiteren Arbeiten ist geplant, die Charakterisierung von Anwendungen bezüglich ihrer Nutzung von L3-Cache und Hauptspeicherbandbreite dafür zu nutzen, CAT nur bei Bedarf an- oder abzuschalten. Dabei ist auch die Frage, welche Partitionsgrößen zum besten Ergebnis führen.

Literaturverzeichnis

[Ba91] Bailey, David H; Barszcz, Eric; Barton, John T; Browning, David S; Carter, Robert L; Dagum, Leonardo; Fatoohi, Rod A; Frederickson, Paul O; Lasinski, Thomas A; Schreiber, Rob S et al.: The NAS parallel benchmarks. The International Journal of Supercomputing Applications, 5(3):63–73, 1991.

[BL16] Blanche, Andreas de; Lundqvist, Thomas: Terrible Twins: A Simple Scheme to Avoid Bad Co-Schedules. In (Trinitis, Carsten; Weidendorfer, Josef, Hrsg.): Proceedings of the 1st COSH Workshop on Co-Scheduling of HPC Applications. S. 25, Jan 2016.

[BM10] Bhadauria, Major; McKee, Sally A.: An Approach to Resource-aware Co-scheduling for CMPs. In: Proceedings of the 24th ACM International Conference on Supercomputing. ICS '10, ACM, New York, NY, USA, S. 189–199, 2010.

[BW17] Breitbart, Jens; Weidendorfer, Josef: Co-Scheduling of HPC Applications. IOS Press, Amsterdam, The Netherlands, Kapitel Detailed Application Characterization and Its Use for Effective Co-Scheduling, S. 69–94, 2017.

[BWT15] Breitbart, Jens; Weidendorfer, Josef; Trinitis, Carsten: Case Study on Co-Scheduling for HPC Applications. In: International Workshop on Scheduling and Resource Management for Parallel and Distributed Systems (SRMPDS 2015). Beijing, China, 2015.

[BWT16] Breitbart, Jens; Weidendorfer, Josef; Trinitis, Carsten: Automatic Co-scheduling based on Main Memory Bandwidth Usage. In: Proceedings of the 20th Workshop on Job Scheduling Strategies for Parallel Processing (JSSPP 2016). Chicago, US, 2016.

[CME16] Clauss, Carsten; Moschny, Thomas; Eicker, Norbert: Dynamic Process Management with Allocation-internal Co-Scheduling towards Interactive Supercomputing. In (Trinitis, Carsten; Weidendorfer, Josef, Hrsg.): Proceedings of the 1st COSH Workshop on Co-Scheduling of HPC Applications. S. 13, Jan 2016.

[dBL17] de Blanche, Andreas; Lundqvist, Thomas: Disallowing Same-program Co-schedules to Improve Efficiency in Quad-core Servers. In (Clauss, Carsten; Lankes, Stefan; Trinitis, Carsten; Weidendorfer, Josef, Hrsg.): Proceedings of the Joined Workshops COSH 2017 and VisorHPC 2017. S. 13–20, Jan 2017.

[Ha14] Haritatos, Alexandros-Herodotos; Goumas, Georgios; Anastopoulos, Nikos; Nikas, Konstantinos; Kourtis, Kornilios; Koziris, Nectarios: LCA: A Memory Link and Cache-aware Co-scheduling Approach for CMPs. In: Proceedings of the 23rd International Conference on Parallel Architectures and Compilation. PACT '14, ACM, New York, NY, USA, S. 469–470, 2014.

[Ha16] Haritatos, Alexandros-Herodotos; Nikas, Konstantinos; Goumas, Georgios; Koziris, Nectarios: A resource-centric Application Classification Approach. In (Trinitis, Carsten; Weidendorfer, Josef, Hrsg.): Proceedings of the 1st COSH Workshop on Co-Scheduling of HPC Applications. S. 7, Jan 2016.

[Mc95] McCalpin, John D: A survey of memory bandwidth and machine balance in current high performance computers. IEEE TCCA Newsletter, 19:25, 1995.

[Pa17] Papadakis, Ioannis; Nikas, Konstantinos; Karakostas, Vasileios; Goumas, Georgios; Koziris, Nectarios: Improving QoS and Utilisation in modern multi-core servers with Dynamic Cache Partitioning. In (Clauss, Carsten; Lankes, Stefan; Trinitis, Carsten; Weidendorfer, Josef, Hrsg.): Proceedings of the Joined Workshops COSH 2017 and VisorHPC 2017. Stockholm, Sweden, S. 21–26, Jan 2017.

[PBL17] Pickartz, Simon; Breitbart, Jens; Lankes, Stefan: Co-scheduling on Upcoming Many-Core Architectures. In (Clauss, Carsten; Lankes, Stefan; Trinitis, Carsten; Weidendorfer, Josef, Hrsg.): Proceedings of the Joined Workshops COSH 2017 and VisorHPC 2017. Stockholm, Sweden, S. 27–32, Jan 2017.

[Su16] Sueß, Tim; Döring, Nils; Gad, Ramy; Nagel, Lars; Brinkmann, André; Feld, Dustin; Schricker, Eric; Soddemann, Thomas: Impact of the Scheduling Strategy in Heterogeneous Systems That Provide Co-Scheduling. In (Trinitis, Carsten; Weidendorfer, Josef, Hrsg.): Proceedings of the 1st COSH Workshop on Co-Scheduling of HPC Applications. S. 37, Jan 2016.

[WB16] Weidendorfer, Josef; Breitbart, Jens: Detailed Characterization of HPC Applications for Co-Scheduling. In: Proceedings of the first Workshop on Co-Scheduling of HPC Applicat ions (COSH 2016). Prague, Czech Republic, 2016.