GeNiuS

Newsletter von GNS und GNS Systems

Ausgabe 1/2014

Animator4, Version 2.1

Seit den Anfangstagen von Animator3 ist die Rechenleistung von Grafikkarten stetig gestiegen. Dank extremer Spezialisierung haben die Prozessoren aktueller Grafikkarten (GPUs) weit mehr Rechenleistung als die Haupt-Prozessoren der Workstation-Computer (CPUs).
Es bietet sich daher an, bestimmte Rechenaufgaben auf die Grafikkarte auszulagern. Bisher beschränkte sich Animator4 auf kleinere Optimierungen. Doch ab Version 2.1 werden der Speicher und der Prozessor der Grafikkarte intensiv genutzt. Das bringt für den Anwender einige Veränderungen mit sich.

Geschwindigkeit!

Animator4 kann 3D-Modelle nun spürbar flüssiger verschieben, drehen, zoomen und animieren. Es schont einfach die Nerven, wenn das angezeigte Modell flüssig auf Benutzereingaben reagiert, statt asynchron herumzuruckeln.
In Benchmarks konnten bis zu 700% Geschwindigkeitszuwachs gemessen werden. Realistisch sollten in den meisten Anwendungsfällen noch immer 50-200% Geschwindigkeitsvorteil sein.

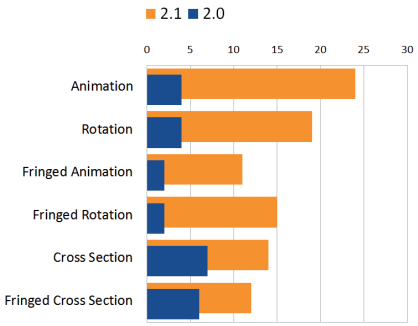


Abb. 1: GeForce GTX 770 (Linux)

Die *Quadro 2000* (Abb. 2) ist eine drei Jahre alte Mid-Range-Grafikkarte. Sie kann stellvertretend für viele Workstations stehen, die gerade im Einsatz sind. Der Leistungszuwachs ist beachtlich.

Wie groß der Geschwindigkeitsgewinn ist, hängt von den Aufgaben ab. Es gibt deshalb noch immer einzelne Anwendungsfälle, in denen sich kein spürbarer Vorteil ergibt. Als Beispiel kann man die Cross-Section Darstellung auf einer Quadro 2000 betrachten (Abb. 2).
Wie man andererseits am Beispiel der leistungsfähigeren *GeForce GTX 770* erkennen kann (Abb. 1), erzielen High-End-Grafikkarten auch bei komplexen Aufgaben wie der Darstellung von Cross Sections einen deutlichen Geschwindigkeitsgewinn.

Viel bringt viel

Frühere Animator-Versionen beanspruchten Grafikkarten kaum. Deshalb war es relativ gleichgültig, ob eine High-End- oder Standard-Grafikkarte im Rechner steckte. Lediglich ein ausgereifter OpenGL-Treiber war Pflicht.

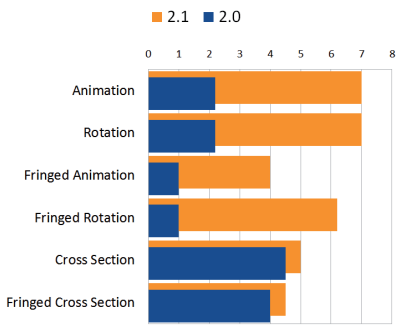


Abb. 2: Quadro 2000 (Windows)

News

Beiträge

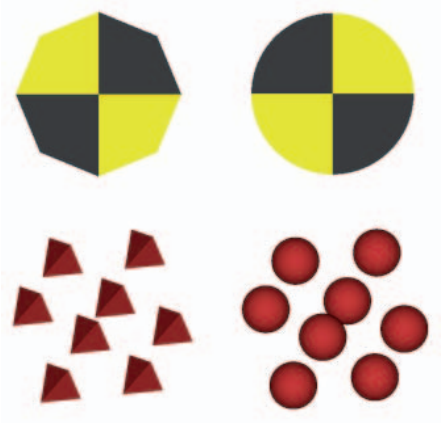
Mit Version 2.1 ändert sich das. High-End-Grafikkarten bringen nun einen viel stärkeren Geschwindigkeitsvorteil. Da in vielen Arbeitsplatzrechnern schon leistungsstarke Grafikkarten verbaut sind, bringen die Änderungen in *Animator4* 2.1 vielen Anwendern quasi gratis einen enormen Leistungszuwachs.

In gleicher Weise profitiert *Animator4* ab Version 2.1 nun von mehr Speicher auf der Grafikkarte. Je mehr Modelldaten direkt auf der Grafikkarte gespeichert werden können, desto seltener müssen Daten nachgeladen werden.

Aber selbst bei Standard-Grafikkarten überwiegen die Vorteile, so dass kein Leistungseinbruch zu befürchten ist.

Kleinigkeiten

Auch Intel Grafikkarten (z.B. *HD 4600*) kommen mit der neuen Technik besser klar und sind deutlich schneller. Dadurch verbessert sich die Leistung auf Laptops mit in die CPU integrierter Grafik (Abb. 3). Optisch erlaubt die neue Technik etwas Feinschliff: Impact-Points sind jetzt rund und nicht mehr 8-eckig. SPH-Elemente sind ebenfalls rund statt Tetraeder (siehe unten).



Das ist erst der Anfang

Die flüssige Darstellung von 3D Modellen ist ein wesentliches Merkmal von *Animator4*. Gerade deshalb ist das Programm bei vielen Anwendern beliebt.

Das Entwicklerteam von Animator4 wird deshalb weiter nach Möglichkeiten suchen, aktuelle Hardware optimal zu nutzen. Wie gewohnt ist diese neue

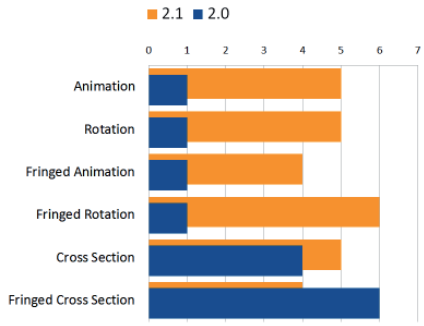


Abb. 3: HD 4600 (Windows)

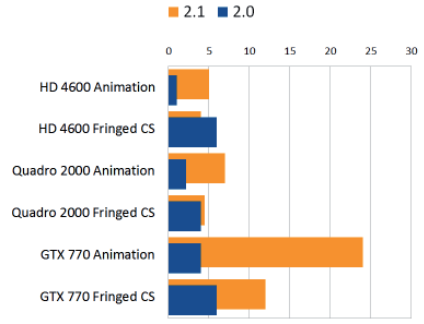


Abb. 4: Grafikkartenvergleich

Version ein Meilenstein, aber sicher kein Endpunkt.

Der technische Hintergrund

Ein Tischler braucht eine Stunde, um einen Stuhl zu schreinern. Wie lange brauchen 100 Tischler für einen Stuhl? Die realistische Antwort lautet sicher nicht 36 Sekunden. Das veranschaulicht einige Probleme der Parallelisierung. Manche Aufgaben lassen sich besser parallelisieren als andere.

Die Darstellung von 3D Modellen gehört zu einfach parallelisierbaren Problemen. Deshalb sind heutige GPUs hochspezialisierte Parallel-Prozessoren (die Nvidia K5000 hat über 1500 Prozessor-Kerne). Man braucht aber auch weiterhin CPUs, die auch schlechter parallelisierbare Aufgaben schnell bewältigen. Manche Aufgaben können sehr effizient an die GPU delegiert werden. Andere liegen im Grenzbereich und werden auf GPU und CPU etwa gleich schnell ausgeführt.

Eine weitere Einschränkung besteht im Datentransfer. Alle nötigen Modelldaten müssen auf die Grafikkarte kopiert werden, weil der vergleichsweise langsame Zugriff auf den allgemeinen Hauptspeicher (RAM) die Geschwindigkeitsvorteile weitgehend zunichte machen würde. Die GPU kann viel schneller auf den eigenen Grafikkarten-Speicher zugreifen als auf das allgemeine RAM. Schließlich müssen die Grafikkarten-Treiber die Ressourcen der Grafikkarte effizient und fehlerfrei verwalten.
Daraus ergeben sich die Anforderungen an die Grafikkarte: ausgereifte OpenGL-Treiber, eine schnelle GPU und genug Speicher auf der Grafikkarte, z.B. eine *Nvidia K4000*.

Der (kleine) Haken

Mehr Leistung für's gleiche Geld. Die teure Grafikkarte tut endlich was für ihr Geld. Wo ist der Haken? Einen kleinen Haken gibt es doch noch: *Animator4 2.1* benötigt eine Grafikkarte, die mindestens *OpenGL 3.3* beherrscht. Das ist im Bereich CAE eigentlich seit Jahren eine Selbstverständlichkeit. Allerdings konnte man *Animator4* bisher auch ohne Grafikkarte auf einem Großrechner laufen lassen. Das geht nun vorerst nicht mehr. Wir arbeiten daran.

Sascha Kremers, GNS mbH
animator@gns-mbh.com

OpenForm - Offen in jeder Form

In der blechverarbeitenden Industrie ist die numerische Simulation als ein den Prozess der Produktentstehung begleitendes Werkzeug nicht mehr wegzudenken. Sowohl in der frühen Planungsphase eines Produktes als auch während seiner Entwicklung und Herstellung findet die numerische Simulation ihren Einsatz, sei es um die Herstellbarkeit abzusichern, die Betriebsmittel zu planen oder die geforderten Eigenschaften des Endproduktes zu garantieren. Je nach Fertigungsprozess, Einsatzgebiet und Anforderung an das Endprodukt gibt es eine Vielzahl an zumeist spezialisierten Softwareprodukten, um die einzelnen Glieder der Produktentwicklungskette zu simulieren.

Jedes dieser Softwareprodukte hat in der Regel seine eigene Anwenderschnittstelle (GUI: Graphical-User-Interface), seine eigene Nomenklatur für die Beschreibung der zu lösenden Aufgabe und sein eigenes Eingabe- und Ausgabeformat. Die numerische Simulation einer längeren Prozesskette ist somit in der Praxis sehr umständlich und zeitraubend und erfordert den erfahrenen Umgang mit verschiedenartiger Simulationssoftware.

Aus dem oben Gesagten ergibt sich der Wunsch, über eine Anwenderschnittstelle (GUI) zu verfügen, in der - unabhängig von der speziellen Nomenklatur einzelner Softwarelösungen - beliebig komplexe Prozessketten definiert und in Form einer allgemeinen Prozesssprache exportiert werden können. Dies ist die Vision hinter OpenForm!

OFPL – OpenForm Process Language

Die interne Darstellung der in OpenForm für die numerische Simulation definierten Fertigungsprozesse erfolgt durch die sogenannte iOFPL (internal OpenForm Process Language). Die iOFPL beschreibt dabei nur den physikalischen Prozess, der später numerisch simuliert werden soll. Mit ihr werden keine Informationen beschrieben, die mit der numerischen Simulation dieser Prozesse in Zusammenhang stehen. Diese Informationen werden in OpenForm separat abgelegt. Neben dem direkten Export von Eingabedateien für einzelne Simulationsprogramme (z.B. INDEED, LS-DYNA, PAM-STAMP, NASTRAN, PAM-CRASH) bietet OpenForm die Möglichkeit, die interne Prozesssprache in Form einer von Menschen lesbaren (ASCII-Format) Metasprache zu exportieren. Diese Metasprache wird OFPL (OpenForm Process Language) genannt. Sie ist die „äußere" Repräsentation der iOFPL und basiert auf Arbeiten der BMW AG. Im Rahmen einer Kooperation zwischen der BMW AG und GNS wurde diese Sprache jedoch erweitert.

Die OFPL ermöglicht die einfache und standardisierte Beschreibung verschiedenartiger Prozesse, die im Laufe einer Produktentwicklung mittels numerischer Simulation abgesichert werden sollen. Diese Beschreibung ist unabhängig von der Simulations-Software, die eingesetzt

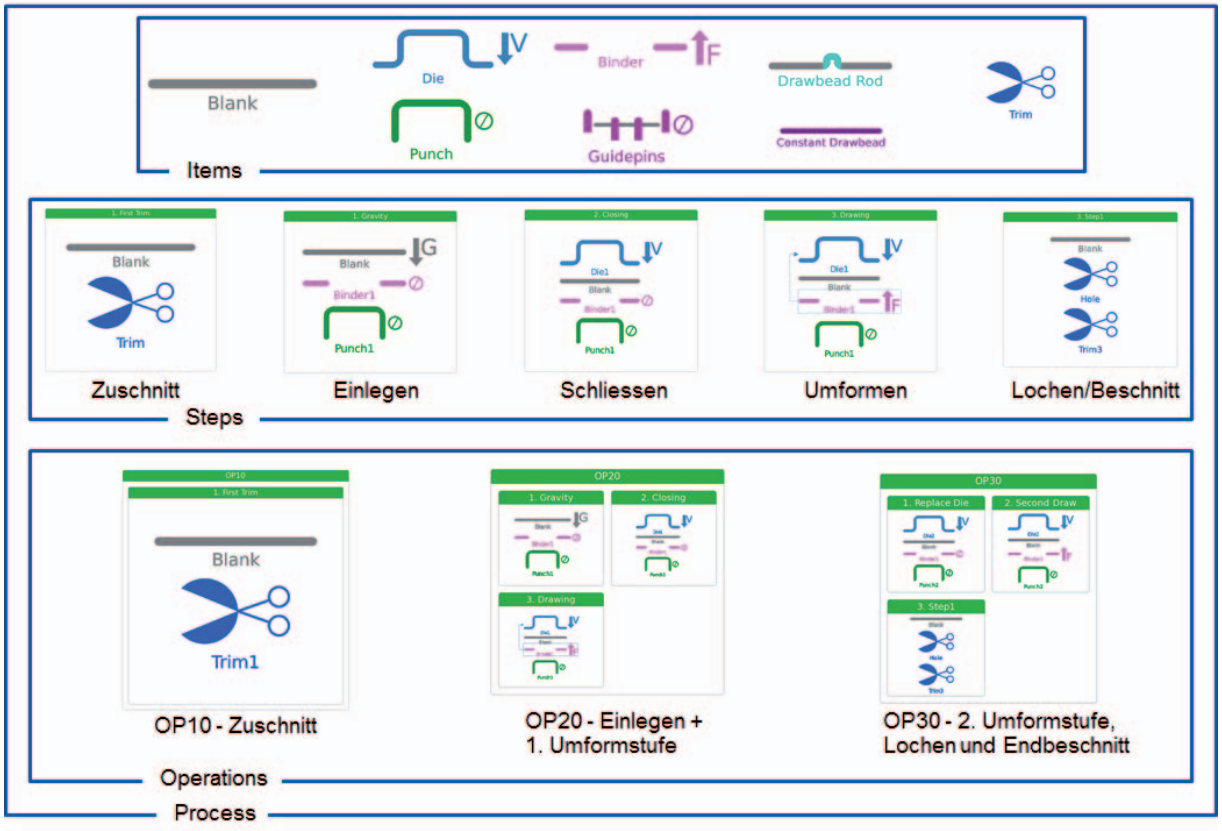


Abb. 1: Hierarchischer Aufbau eines Tiefziehprozesses (Items/Steps/Operations)

werden soll. Mithilfe geeigneter Konverter können daraus Eingabedateien für beliebige Simulationsprogramme generiert werden.

Wird die Metasprache OFPL anstelle der in OpenForm zur Verfügung stehenden Direktschnittstellen (Konverter) zu einzelnen Simulationsprogrammen verwendet, so lässt sich zusätzlich der Vorteil einer redundanzfreien Datenarchivierung nutzen. Außerdem sind die zu archivierenden Datenmengen von vergleichsweise geringem Umfang.

Aufbau eines Prozesses

OpenForm bietet dem Anwender eine hierarchische Vorgehensweise bei der Aufbereitung von Prozessbeschreibungen für die numerische Simulation eines Fertigungsprozesses. So steht dem Anwender auf der untersten Ebene eine umfangreiche Bibliothek an Grundelementen (Items) zur Verfügung, mit denen - einfach und flexibel - einzelne Prozessschritte beschrieben, diese zu Prozessoperationen verknüpft und schließlich diese wiederum zu einem Prozess aus einer komplexen Prozesskette zusammengestellt werden können (Abb. 1). So kann man aus einer Fülle von Umformelementen wie z.B. Rohlingen, Werkzeugen, Ziehleisten, Einweisern, Wirkmedien usw. einzelne Schritte eines Umformprozesses zusammenstellen wie z.B. die Schritte Zuschnitt des Rohlings/Werkstücks, Einlegen ins Werkzeug, 1. Umformstufe, Lochen, 2. Umformstufe, Endbeschnitt usw.

Diese Schritte werden dann zu Operationen zusammengefasst:
OP10 beinhaltet z.B. den Schritt Zuschnitt, OP20 die Schritte Einlegen und 1. Umformstufe, OP30 die 2. Umformstufe, das Lochen und den Endbeschnitt.

Diese Prozessschritte zusammen könnten dann z.B. einen Tiefzieh- oder IHU-Prozess beschreiben. Ein solcher Umformprozess stellt dann einen Baustein in einer komplexen Produktfertigungskette dar.

Beispiel einer Prozesskette

Basierend auf der internen solver-unabhängigen Prozesssprache iOFPL bietet OpenForm eine einheitliche Bedienoberfläche, mit der entweder über die zur Verfügung stehenden Direktschnittstellen oder aber über die Metasprache OFPL im Zusammenhang mit externen Konvertern verschiedenartige Anwendungen bedient werden können. Dadurch können komplexe Prozessketten für die numerische Simulation aufbereitet, diese bei Vorhandensein entsprechender automatisierter Workflows durchgeführt und anschließend deren Ergebnisse in OpenForm ausgewertet werden. So ließe sich z.B. - nach einer entsprechenden Aufbereitung der einzelnen Bausteine - für den Herstellungsprozess eines tiefgezogenen Trägers aus z.B. flexibel gewalzten Platinen (TRB) der Walzvorgang, die Umformung, das Fügen bis hin zur Bauteilbeanspruchung unter Betriebslast oder im Crash in Form einer durchgehenden Berechnungskette simulieren.

In Abbildung 2 ist exemplarisch der Aufbau einer Teilprozesskette mit OpenForm abgebildet, bei der unterschiedliche Simulationsprogramme zum Einsatz kommen können (z.B. INDEED, NASTRAN und PAM-CRASH).

Fazit und Ausblick

Mit der Software OpenForm ist ein erster großer Schritt realisiert worden, die numerische Simulation von Herstellungsprozessen der Fertigungsindustrie zu vereinheitlichen und somit zu vereinfachen.

Basierend auf der OpenForm eigenen Prozesssprache iOFPL, können verschiedene Prozesse unabhängig von der später eingesetzten numerischen Software beschrieben und zu komplexen Prozessketten zusammengestellt werden. Mit Hilfe der in OpenForm zur Verfügung stehenden Schnittstellen können die Inputdecks der einzelnen Prozesse für die jeweilig passende Software erzeugt und die Simulation angestoßen werden. Im Anschluss können die verschiedenen Berechnungsergebnisse in OpenForm visualisiert und praxisbezogen ausgewertet werden.

Koutaiba Kassem, GNS mbH
openform@gns-mbh.com

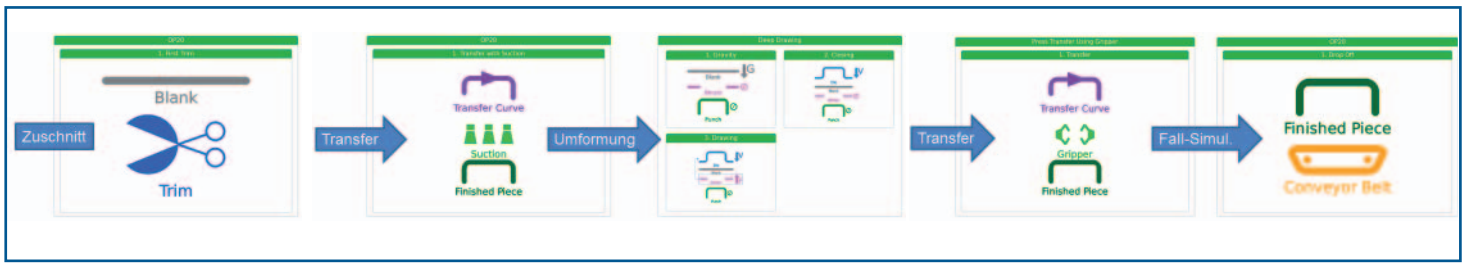


Abb. 2: Beschreibung einer Teil-Prozesskette (Zuschnitt, Transfer-, Umform- und Fallsimulation)

Linux Ressourcenmanagement mit Control Groups

Kontrollgruppen, Control Groups oder kurz „cgroups“ sind eine 2007 eingeführte Funktionalität des Linux-Kernels. Mittels cgroups ist es möglich, Prozesse zur Laufzeit in einheitliche hierarchische Gruppen einzuordnen. Den Prozessen dieser Gruppen kann man über den Kernel spezielles Verhalten bezüglich ihrer Ressourcennutzung zuteilen. Die Einrichtung und Konfiguration der cgroups im Kernel erfolgt über das virtuelle „cgroup“ Dateisystem.

Eine Nutzung von *cgroups* bietet sich an, um eine Priorisierung, ein Accounting, Isolation oder eine Limitierung unterschiedlicher Prozessgruppen vorzunehmen. Dadurch ist es möglich, Service Level Agreements (SLAs) umzusetzen oder mehrere virtuelle Instanzen voneinander zu isolieren. Bei modernen Linux Distributionen werden *cgroups* automatisch angelegt und verwendet. Sogar Android verwendet *cgroups*.

Die vier wichtigsten Begriffe im Umgang mit *cgroups* sind die *Hierarchie*, das *Subsystem*, die *cgroup* und der *task*. Am einfachsten lassen sich die Begriffe direkt durch ihre Repräsentation im virtuellen „*cgroup*“ Dateisystem verstehen.

Ein Dateisystem vom Typ „cgroup“, repräsentiert die sogenannte *Hierarchie*. Es kann mit dem Befehl

```
mount -t cgroup -o
cpuset,freezer,memory,blkio
batch_jobs /cgroup/batch_jobs
```

in das vorhandene Verzeichnis

```
/cgroup/batch_jobs
```

eingehängt werden. Die Dateisystemoptionen `cpuset,freezer,memory,blkio` sind die in diesem Beispiel verwendeten Subsysteme. Durch das Erstellen eines Unterverzeichnisses

```
mkdir -p /cgroup/batch_jobs/job1
```

im virtuellen cgroup-Dateisystem wird die cgroup job1 erstellt.

Mit dem Befehl

```
/bin/echo 1234 >> /cgroup/batch_
jobs/job1/tasks
```

wird der Prozess mit der PID (process identifier) `1234` der cgroup „`job1`“ zugeordnet. An der Stelle des Blockdevice „device“ im „mount“-Befehl

```
mount [-t vfstype] [-o options]
device dir
```

steht beim Dateisystemtyp „cgroup“ der Name der *Hierarchie*. Im Beispiel

```
mount -t cgroup -o
cpuset,freezer,memory,blkio
batch_jobs /cgroup/batch_jobs
```

heißt die *Hierarchie* `batch_jobs`. Die *Hierarchie* enthält alle Prozesse des Systems, alle *cgroups* (Unterverzeichnisse) gehören ebenfalls zur Hierarchie.

Die Subsysteme `freezer,memory,blkio` geben vor, welche „Dateien“ sich in der *Hierarchie* befinden. Über eine Interaktion mit diesen „Dateien“ können die Gruppen konfiguriert oder überwacht werden.

Der Mountpoint der *Hierarchie* selbst gibt vor in welchem Verzeichnis die root cgroup repräsentiert wird:

```
/cgroup/batch_jobs
```

cgroups in *Hierarchien* werden durch Unterverzeichnisse in der *root cgroup* repräsentiert. Auch die *cgroups* können weitere Unterverzeichnisse, sogenannte verschachtelte (*nested*) *cgroups*, enthalten.

In jeder *cgroup* befindet sich eine Datei namens „tasks“. Sie bildet die Schnittstelle für die Zuordnung der Prozesse in die *cgroup*. Durch Schreiben einer PID in die „tasks“-Datei einer *cgroup* wird der Prozess der jeweiligen *cgroup* zugeteilt. Laufende Prozesse können aus einer *cgroup* nur entfernt werden, indem sie einer anderen *cgroup* der gleichen *Hierarchie* (z.B. der *root cgroup*) zugeordnet werden.

Ein Leeren der „tasks“-„Datei“ auf Dateisystemebene z.b. durch

```
/bin/true > tasks
```

ist explizit nicht möglich.

Nicht vergebene PIDs lassen sich einer *cgroup* nicht zuordnen:

```
/bin/echo 123 >> tasks
   /bin/echo: write error: No
   such process
```

Regeln:
Im Red Hat Enterprise Linux 6 Resource Management Guide (siehe Quellen) unter „*1.2. Relationships Between Subsystems, Hierarchies, Control Groups and Tasks*“ hat *Red Hat* vier Regeln im Umgang mit *cgroups* zusammengetragen.

1.) Einer Hierarchie muss immer mindestens ein Subsystem zugeordnet sein. Es können auch mehrere Subsysteme in einer Hierarchie verwendet werden.

2.) Ein Subsystem kann aber nicht in zwei Hierarchien verwendet werden wenn mindestens eine davon ein anderes Subsystem nutzen soll.

3.) Nach dem Erstellen einer Hierarchie befinden sich alle Prozesse des Systems in der *root cgroup*. Ein Prozess kann in einer Hierarchie immer nur genau einer *cgroup* zugeordnet werden.

4.) Ein neuer Prozess befindet sich unabhängig davon wie er gestartet wurde (fork, exec, clone) in der *cgroup* seines Elternprozesses. Danach kann er aber in beliebige andere *cgroups* gruppiert werden.

Subsysteme
In *Red Hat Enterprise Linux 6* stehen diese Subsysteme zur Verfügung:

- blkio Block Devices (Festplatte, SSD, ...)
- cpu CPU Scheduling
- cpuacct CPU Accounting
- cpuset CPU Placing
- devices Whitelist Controller
- freezer „Einfrieren“ und „Auftauen“
- memory Speichernutzung: Begrenzen und Überwachen
- net_cls network classid (für den traffic controller (tc))
- net_prio Dynamische Netzwerk Priorisierung
- ns Namespaces (Virtualisierung)

Die Subsysteme *blkio, cpu* und *net_cls* können verwendet werden, um die Leistung von Block Devices, der CPUs oder des Netzwerkes (Traffic Shaping) zwischen *cgroups* zu priorisieren oder generell zu begrenzen. Das net_cls Subsystem vergibt hierbei nur die sog. ClassID, der Linux Traffic Controller (tc) muss zusätzlich verwendet werden um in Abhängigkeit von der ClassID Traffic Shaping oder Drosselung zu implementieren.

Mit den Subsystemen *blkio, cpuacct* und *memory* kann man die Nutzung von „Block Devices“, der CPUs beziehungsweise der Speichernutzung überwachen.

Die „proportional weight division“ des Subsystems *blkio* kann z.B. verwendet werden, um Prozesse in unterschiedliche relative IO-Prioritätsklassen zu gruppieren. Wenn man nun z.B. eine Hierarchie mit dem *blkio* Subsystem erstellt und darin zwei *cgroups* „schnell“ und „langsam“ anlegt, dann kann mit

```
/bin/echo 1000 >> /cgroup/blkio/
schnell/blkio.weight
```

```
/bin/echo  100 >> /cgroup/blkio/
langsam/blkio.weight
```

der cgroup „schnell“ 1000 Anteile an der IO-Geschwindigkeit und der *cgroup* „langsam“ 100 Anteile zugeordnet werden.

Gleichzeitiges Lesen von zwei Prozessen in je einer dieser *cgroups* zeigt ein Lesegeschwindigkeitsverhältnis von ziemlich genau 10:1 (705:70,9 [MB/s]) zwischen Prozessen in der *cgroup* „schnell“ zu Prozessen in der *cgroup* „langsam“.

In Abbildung 1 a und b ist dargestellt, wie groß die Anteile an der IO-Performance ohne cgroups und mit den hier im Beispiel genannten Werten sind.

Der große Vorteil dieses Verfahrens im Vergleich zum Drosseln (`blkio.throttle*`) ist, dass Prozesse jeder *cgroup* die volle IO-Geschwindigkeit des Systems alleine nutzen können, wenn sie alleine laufen. Ein gutes Anwendungsszenario für das *blkio* Subsystem ist das Backup von Daten mit niedriger Priorität um produktive Prozesse nicht zu beeinflussen.

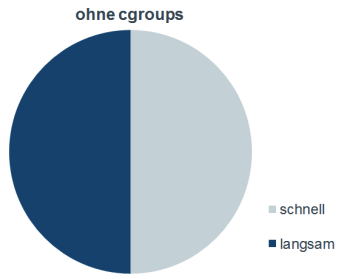


Abb. 1a: Festplattengeschwindigkeitsverhältnisse: ohne cgroups 1:1

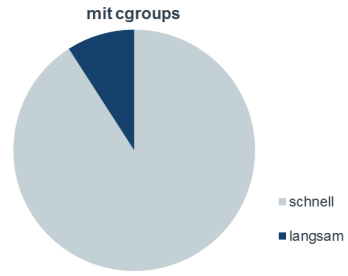


Abb. 1b: Festplattengeschwindigkeits verhältnisse: mit cgroups 10:1

Mittels des Subsystems *„freezer“* lassen sich Prozesse die einer cgroup zugeordnet sind „einfrieren“

```
/bin/echo FROZEN >> ${CGROUP}/
freezer.state
```

und auch wieder „auftauen“:

```
/bin/echo THAWED >> ${CGROUP}/
freezer.state
```

Dabei werden ebenfalls alle Prozesse der *nested cgroups* eingefroren.
Damit ist auch selbsterklärend, dass die *root_cgroup* einer Hierarchie nicht eingefroren werden kann. Im *„cgroup“*-Dateisystem fehlt daher auch die *freezer.state* in der *root cgroup*. „Eingefroren“ bedeutet, dass *„FROZEN“* (gefroren) und *„THAWED“* (getaut) die einzigen erlaubten Schlüsselbegriffe sind, die in die *freezer.state* geschrieben werden können. Beim Lesen der *freezer.state* kann auch `FREEZING` zurück gemeldet werden.

Der Befehl

```
/bin/echo „Hallo Welt“ >> freezer.state
```

liefert diese Fehlermeldung:

```
/bin/echo: write error: Invalid
argument
```

Beim *„freezer“* wurde von Anfang an an High-Performance-Computing (HPC) Umgebungen gedacht. In großen HPC-Umgebungen werden typischerweise lang laufende Jobs für kurz laufende Jobs angehalten, um temporär Maschinen und/oder Softwarelizenzen bereitzustellen und so den Gesamtdurchsatz zu optimieren. Auch ist es möglich, auf „eingefrorene“ Prozesse ein Checkpointing anzuwenden um Restarts zu ermöglichen.

Durch Verwendung des memory Subsystems lassen sich Speicherlimits definieren und die Speichernutzung überwachen. Im Beispiel werden wir beides mehrfach beobachten:

Erstellen einer Test-cgroup:

```
mkdir -p /sys/fs/cgroup/
memory/test
```

Wechseln in die cgroup:

```
cd $_
```

Zuordnung der Shell und ihrer zukünftigen „child"-Prozesse in die cgroup:

```
echo $$ > tasks
```

Prüfung der Speichernutzung der *cgroup* (also nur der Shell):

```
cat memory.memsw.max_usage
_in_bytes
   262144
```

Definieren eines Speicherlimits von 100MiB:

```
echo $((100*1024**2)) >
memory.limit_in_bytes
```

Starten eines Prozesses in der *cgroup* der Shell, der 1G allokiert und belegt:

```
memhog 1G
   ..........Killed
```

Der Prozess wurde vom Kernel terminiert. Die maximale Speichernutzung der *cgroup* hat sich auf exakt 100MiB geändert:

```
cat memory.memsw.max_usage_in_
bytes
  104857600
```

Definieren eines Speicherlimits von 100GiB:

```
echo $((100*1024**3)) > memory.
limit_in_bytes
```

Starten des Prozesses, der 1G allokiert und belegt:

```
memhog 1G
   ..................................................
```

Mit dem neuen Speicherlimit wurde der Prozess nicht terminiert. Die maximale Speichernutzung der *cgroup* hat sich auf etwas über 1GiB geändert:

```
cat memory.memsw.max_usage_in_
bytes
   1074065408
```

Libcgroup

Die *„libcgroup"* stellt eine API für die Nutzung von *cgroups* bereit. Sie enthält die Init-Services *cgconfig* und *cgred,* die sich über Konfigurationsdateien konfigurieren lassen. Durch Nutzung von *cgconfig* (Control Groups Configuration Startup) bleibt die Verfügbarkeit der *„cgroups"* auch nach Neustarts des Systems erhalten. Mit *cgred* (CGroups Rules Engine Daemon) lassen sich Prozesse aufgrund ihres Besitzers (User und/oder Gruppe) oder des Namens ihres Executables gewissen *cgroups* zuteilen.

Zudem werden diverse Befehle, die den Umgang mit *cgroups* vereinfachen sollten durch die *libcgroup* bereitgestellt.

Weitergehende Informationen zur Libcgroup finden sich auf der Projektwebseite *„http://libcg.sourceforge.net/ ".*

Bei modernen Linux Distributionen mit dem Init-Ersatz „systemd" oder „OpenRC" werden die Init-Services automatisch in passend angelegten *cgroups*

```
/sys/fs/cgroup/${SUBSYSTEM}
```

gestartet.

Ein ganz praktischer Vorteil von *cgroups* ist, dass sie auch Lock-Dateien ersetzen können. Der Linux-Kernel protokolliert in den tasks der *cgroup* eigenständig, welche Prozesse zu dem Service gehören und entfernt diese, wenn sie beendet werden.

Release Agent

Mit den Befehlen

```
echo 1 > notify_on_release
echo /tmp/cgroup-release-agent.
   sh > release_agent
```

kann man einen „Release Agent" konfigurieren. Der in release_agent definierte Befehl wird gestartet, wenn der letzte Prozess einer *cgroup* beendet wurde. Der Release Agent kann wie z.B. bei OpenRC genutzt werden, um die *cgroup* zu entfernen:

```
cgroup=/sys/fs/cgroup/openrc
PATH=/bin:/usr/bin:/sbin:/usr/
                              sbin
  if [ -d ${cgroup}/"$1" ]; then
          rmdir ${cgroup}/"$1"
  fi
```

Es ist möglich zuvor, wenn das passende Subsystem vorhanden ist, Accounting-Informationen, CPU-Zeit, Speicher/Swap-Nutzung und IO-Nutzung auszulesen.

CPU- und IO-Performance lassen sich einfach zwischen unterschiedlichen Prozessgruppen aufteilen. Speichergrenzen lassen sich flexibel definieren und überwachen. Bei Bedarf können alle Prozesse einer cgroup gemeinsam angehalten oder wieder aufgeweckt werden.

Mit den *cgroups* wurde ein mächtiges Werkzeug geschaffen, um das Verhalten des Linux Kernels für gewisse Prozessgruppen gezielt konfigurierbar zu machen.

Quellen

- Linux Kernel Dokumentation: https://www.kernel.org/doc/Documentation/cgroups/
- Red Hat Enterprise Linux 6.5 - Resource Management Guide: https://access.redhat.com/site/documentation/en-US/Red_Hat_Enterprise_Linux/6/html/Resource_Management_Guide/index.html
- Libcgroup: http://libcg.sourceforge.net/

Dr. Ramin Torabi, GNS Systems GmbH
ramin.torabi@gns-systems.de

Termine

2015 SIMULIA Community Conference

Datum: 18.-21. Mai 2015
Veranstaltungsort: Berlin

Aussteller: GNS mbH

Bis zum Erscheinen des nächsten GeNiuS können Sie unsere Termine auf den jeweiligen Webseiten von GNS mbH und GNS Systems GmbH erfahren.

GNS mbH
Am Gaußberg 2
38114 Braunschweig
Telefon: 05 31-8 01 12 0
Fax: 05 31-8 01 12 79
www.gns-mbh.com

GNS Systems GmbH
Am Gaußberg 2
38114 Braunschweig

Wir ziehen um,
ab 01.02.2015 finden Sie
GNS Systems GmbH in der
Theodor-Heuss-Str. 5
38122 Braunschweig

Telefon und Fax bleiben wie bisher
Telefon: 05 31-1 23 87 0
Fax: 05 31-1 23 87 11
www.gns-systems.de

Impressum

Ausgabe 1/2014
Erscheinungstermin: Dezember 2014
Herausgeber: GNS Systems GmbH

Verantwortlich: Jan Martini
Redaktion: Anette Tröger
Layout: Anette Tröger

Steigern der Ablaufgeschwindigkeit von langen Animator4 Session Files

Ein großer Teil der Zeit beim Abarbeiten von *Animator4* Session Files wird erfahrungsgemäß durch das Einlesen der Daten verbraucht. Dieses kann man oft durch das Einlesen einer reduzierten Anzahl von Zeitschritten verringern. Bei Kurven sollte man nur ein ‚`rea c2d`' Kommando verwenden, um alle Kurven einzulesen. Hier hilft der Zeilenumbruch ‚`@@`', den ‚`rea c2d`' Befehl übersichtlich zu halten.

Bei langen Session Files reduziert sich der Anteil, den das Einlesen der Daten braucht, zunehmend im Verhältnis zu den anderen Kommandos. Es müssen nun andere Optimierungen her, welche zu einer Steigerung der Ablaufgeschwindigkeit führen.

Der Befehl ‚`vie upd man`' verhindert unnötiges Neuzeichnen der Fensterinhalte. Er entspricht dem früher gebräuchlichen ! vor den Befehlen.

Durch Abschalten der Toolbars: ‚`xcm too off`' und dem Kommandofenster: ‚`xcm dia com off`' verhindert man unnötige Aktualisierungen dieser GUI Elemente.

Soll eine Präsentation erstellt werden, kann mit dem Befehl ‚`opt pre udo num X`' die Anzahl der Undo-Schritte vorgegeben werden. Dieses spart nicht nur Hauptspeicher, sondern auch Zeit.

Verwendete Pattern werden bei jedem Neuzeichnen aktualisiert. Dies lässt sich mit ‚`set pat upd man`' verhindern. Wird es am Ende des Session Files wieder eingeschaltet, so geschieht die Aktualisierung nur einmal.

Das Laden eines Layouts (Global options -> Gui -> Startup) kann nicht nur ungewollte Nebeneffekte haben, sondern auch Zeit kosten, wenn z. B. aktivierte ‚Track' Knöpfe abgespeichert wurden. Hier bietet es sich an, um einen definierten Ablauf zu gewährleisten, das Kommandozeilenargument -u zu benutzen. Dies gibt ein Verzeichnis vor, aus dem die Benutzereinstellungen gelesen werden. Man kann dann auch die Datei *options.ses* (lädt Standardwerte für die opt Kommandos) kürzer gestalten.

Abstandsberechnungen zwischen zwei Gruppen können sehr viel Zeit brauchen. Hier empfielt es sich die Elementgruppen auf das Nötigste (z.B. Außenhaut eines Dummys) zu beschränken. Auch sollte man erst den auszuwertenden Zeitschritt wählen, bevor man den ‚`ide dst gro ...`' Befehl benutzt.

Benötigt man Kurven, welche sich mit dem Befehl ‚`ide his ...`' aus dem 3D-Model ergeben, so kann man mit der Option ‚`opt iht lis`' verhindern, dass für die Kurven jeweils eigene Fenster erzeugt werden, die man vielleicht gar nicht benötigt.

Carsten Thunert, GNS mbH
animator@gns-mbh.com