EMB² = Parallel + Heterogen

Parallele Programmierung von Systems-on-a-Chip

Tobias Schüle, Siemens AG

Die Embedded Multicore Building Blocks (EMB²) sind eine als Open Source Software zur Verfügung stehende Bibliothek für die parallele Programmierung von eingebetteten Systemen. EMB² basiert auf MTAPI (Multicore Task Management API), einem Standard für das Task-Management in mit C/C++ implementierten Applikationen. Im Folgenden geben wir einen Überblick über EMB² und zeigen auf, wie sich Parallelität über die Grenzen von klassischen Multicore-Prozessoren hinaus mittels MTAPI nutzen lässt.

Bei der Programmierung eingebetteter Systeme sind Entwickler heutzutage oftmals mit Parallelität und Heterogenität konfrontiert [1]. Parallelität ist auf Applikationsebene erforderlich, um die Leistung von Multicore-Prozessoren ausschöpfen zu können. Dies ist jedoch mit zahlreichen Hürden und Fallstricken bei der Software-Entwicklung verbunden. Erschwerend kommt hinzu, dass viele eingebettete Systeme neben mehreren, gleichartigen Prozessorkernen spezielle Beschleuniger wie Signalprozessoren oder gar programmierbare Logikbausteine (FPGAs) enthalten. Tatsächlich bestehen moderne Systems-on-a-Chip (SoCs) aus verschiedensten Prozessoren, die für unterschiedliche Zwecke optimiert sind. Solche SoCs zeichnen sich durch eine hohe Leistung bei relativ geringem Energieverbrauch aus. Zu den Kehrseiten gehören eine hohe Komplexität der Software-Entwicklung sowie Herstellerabhängigkeit und damit mangelnde Portabilität. Der MTAPI-Standard [2] verspricht, diese Probleme zu lösen. Bevor wir jedoch auf die grundlegenden Konzepte von MTAPI eingehen, geben wir einen Überblick über EMB².

Embedded Multicore Building Blocks (EMB²)

Abbildung 1 zeigt die wesentlichen Bibliotheksbausteine und deren Einordnung im Gesamtsystem. Auf unterster Ebene befindet sich die Basisbibliothek, die vom Betriebssystem und der Prozessorarchitektur abstrahiert. Darauf baut die MTAPI-Implementierung auf, die entweder direkt von der Applikation aus nutzbar ist oder indirekt über die Komponenten *Algorithms* und *Dataflow*. Letztere stellen häufig benutzte, parallele Algorithmen beziehungsweise Schablonen für die Verarbeitung von Datenströmen zur Verfügung. Eine Besonderheit der MTAPI-Implementierung ist die Unterstützung von Task-Prioritäten und -Affinitäten. Diese unterstützen die Umsetzung von Echtzeiteigenschaften und erlauben eine feingranulare Kontrolle über die Hardware. So können beispielsweise mittels Affinitäten Prozessorkerne für bestimmte Aufgaben reserviert werden.

Desweiteren stehen dem Entwickler threadsichere Datenstrukturen (Container) zur Verfügung, die speziell für den Einsatz in eingebetteten Systemen ausgelegt sind. Dazu gehört zum Einen, dass die Datenstrukturen während des Betriebs keinen dynamischen Speicher allokieren – für viele eingebettete Systeme, insbesondere in sicherheitskritischen Bereichen, ein Muss. Zudem kommen sie ohne blockierende Synchronisationsmechanismen aus (*lock-/wait-free*), woraus sich Garantien bzgl. des Fortschritts der zugreifenden Threads ableiten lassen [3]. Dies ist gerade in eingebetteten Systemen ein nicht zu unterschätzender Vorteil gegenüber klassischen, blockierenden Verfahren.

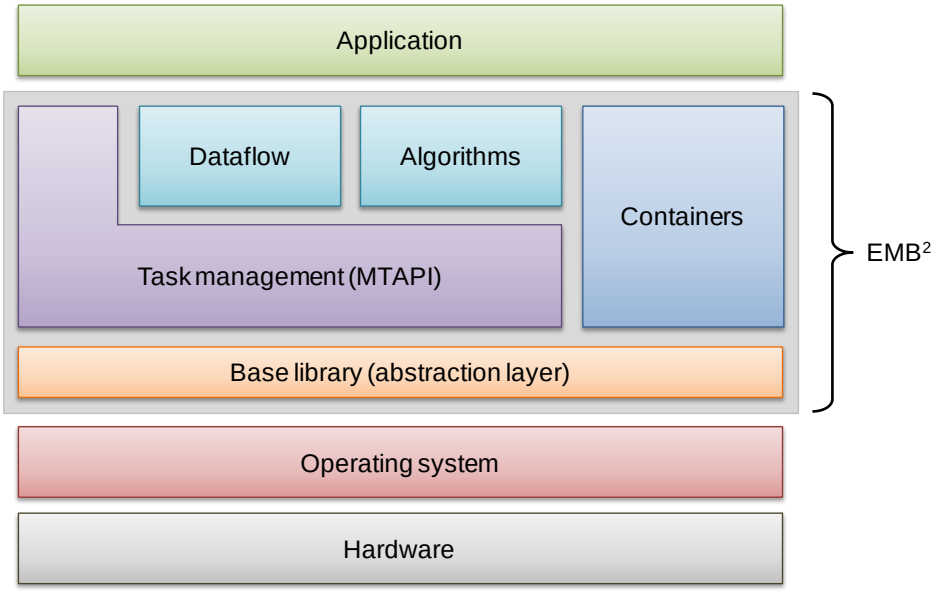

Abb. 1: Überblick über die Embedded Multicore Building Blocks (EMB²)

Eine ausführlichere Beschreibung der Komponenten mit Code-Beispielen ist in [4] und [5, 6] zu finden (siehe dazu https://github.com/siemens/embb/).

Multicore Task Management API (MTAPI)
Um die Leistung von Multicore-Prozessoren ausnutzen zu können, muss die zu verrichtende Arbeit in kleine Teile (*Tasks*) zerlegt werden. Da Threads für diesen Zweck in der Regel zu schwergewichtig sind, wurden in den letzten ein bis zwei Jahrzehnten effiziente Scheduling-Verfahren entwickelt, die ausführbereite Tasks auf die vorhandenen Kerne abbilden. Mit MTAPI stehen seit einigen Jahren auch standardisierte Schnittstellen für die Erzeugung, Verwaltung und Synchronisation von Tasks zur Verfügung. Wesentliche Ziele bei der Definition von MTAPI waren die Benutzbarkeit in kleinen, ressourcenbegrenzten Systemen über eine leichtgewichtige API sowie die Unterstützung von heterogenen Systemen mit verteiltem Speicher und unterschiedlichen Befehlssatzarchitekturen. Zudem setzt MTAPI nicht notwendigerweise ein Betriebssystem voraus, sondern kann auch direkt auf der Hardware (*bare metal*) betrieben werden.

Abbildung 2 illustriert die Programmierung von heterogenen Systemen mit MTAPI. Das Beispielsystem besteht aus einer Multicore-CPU mit vier Kernen, wobei ein Kern eine besondere Rolle mit eigenem Betriebssystem einnimmt, einem Graphikprozessor (GPU) und einem Signalprozessor (DSP). Diese Komponenten werden jeweils durch einen MTAPI-Knoten (*Node*) repräsentiert und bilden zusammen eine Domäne (*Domain*). Als Benutzer kann man entweder explizit angeben, auf welchem Knoten ein Task ausgeführt werden soll, oder die Entscheidung darüber dem Scheduler überlassen. EMB² verwendet standardmäßig für das Scheduling innerhalb eines Knotens prioritätsbasiertes *Work Stealing*, während innerhalb einer Domäne Tasks in Abhängigkeit der Knotenauslastung verteilt werden. MTAPI selbst gibt jedoch keine Scheduling-Verfahren vor und so ist es auch in EMB² möglich, eigene Scheduling-Algorithmen einzusetzen.

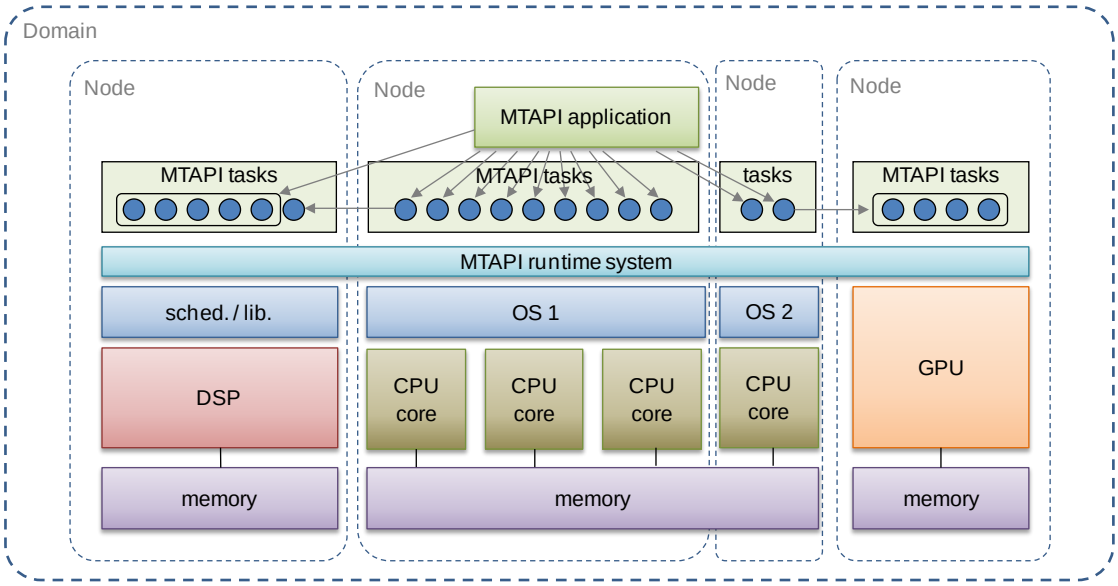

Abb. 2: Beispiel für die Programmierung von heterogenen Systemen mit MTAPI

Neben Tasks kennt MTAPI zwei weitere, damit verwandte Konzepte: *Jobs* und *Actions*. Ein Task ist gemäß MTAPI-Spezifikation die Ausführung eines Jobs mit bestimmten Daten. Ein Job wiederum repräsentiert die auszuführende Funktion auf einer abstrakten Ebene. Jeder Job wird durch mindestens eine Action implementiert. Auf diese Weise können in heterogenen Systemen Implementierungen für verschiedene Prozessoren hinterlegt werden. Eine Action kann sogar komplett in Hardware implementiert sein. Abbildung 3 verdeutlicht den Zusammenhang zwischen Tasks, Jobs und Actions. Da Actions sehr unterschiedlicher Natur sein können, stellt EMB² eine Plugin-Schnittstelle zur Verfügung, über die benutzerdefinierte Actions in MTAPI integriert werden können. So kann zum Beispiel eine FPGA-basierte Action mittels Plugin einem Job zugeordnet und transparent zu anderen Actions ausgeführt werden. Neben der Unterstützung heterogener Systeme hat die Trennung von Jobs und Actions den Vorteil, dass sich damit Applikationen für eine aus unterschiedlichen Hardware-Konfigurationen bestehende Produktfamilie auf einheitliche Weise entwickeln lassen.

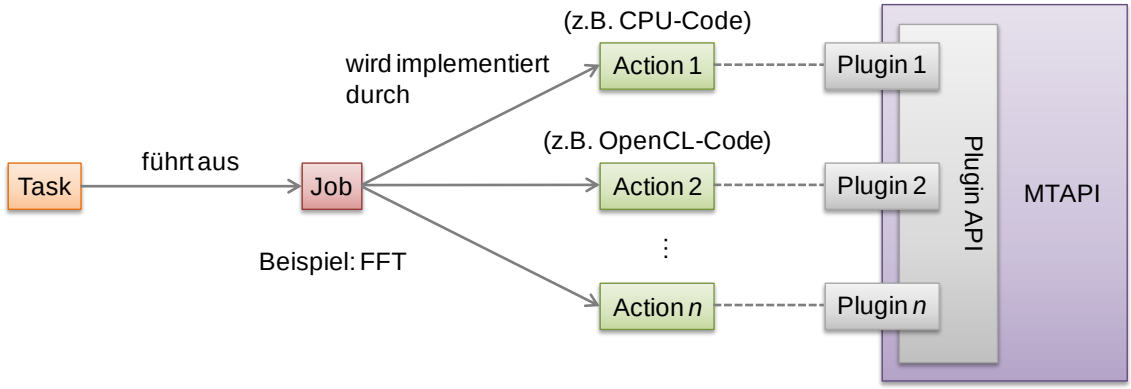

Abb. 3: MTAPI-Konzepte (Task, Job, Actions) und Plugins

EMB² stellt vordefinierte Plugins für häufig benutzte Technologien zur Verfügung. Das folgende Beispiel zeigt die Erzeugung einer Action für OpenCL-Code. Zunächst wird die entsprechende Header-Datei eigebunden, der OpenCL-Kernel definiert und anschließend das Plugin initialisiert:

```
#include <embb/mtapi/c/mtapi_opencl.h>
const char * kernel = "__kernel void MyKernel(…) {…}\n";
mtapi_status_t status;
mtapi_opencl_plugin_initialize(&status);
```

Nun kann die Action registriert und `OPENCL_JOB` zugeordnet werden:

```
float node_local = 1.0f;
action = mtapi_opencl_action_create(
  OPENCL_JOB,
  kernel, "MyKernel", local_work_size, element_size,
  &node_local, sizeof(float),
  &status);
```

Dabei sind `local_work_size` und `element_size` OpenCL-spezifische Argumente, und `node_local` verweist auf gemeinsam genutzte Daten innerhalb des Knotens [5, 6]. Bevor der Job durch einen Task ausgeführt wird, können ihm optional weitere Actions zugeordnet werden [2].

Zusammenfassung

Die parallele Programmierung gewinnt auch bei der Entwicklung eingebetteter Systeme zunehmend an Bedeutung, da von Singlecore-Prozessore keine wesentliche Geschwindigkeitssteigerung mehr zu erwarten ist. Herkömmliche threadbasierte Ansätze sind jedoch oft fehleranfällig, ineffizient und umständlich zu benutzen. Eine weitere Herausforderung bilden heterogene Systems-on-a-Chip, die verschiedenartige Prozessoren auf einem Chip vereinigen. Standardisierte Schnittstellen wie MTAPI helfen, von dieser Diversität zu abstrahieren und Parallelität auf Systemebene effizient zu nutzen.

Literaturverzeichnis

[1] H. Alkhatib, P. Faraboschi, E. Frachtenberg, H. Kasahara, D. Lange, P. Laplante, A. Merchant, D. Milojicic, K. Schwan. “IEEE CS 2022 Report”. IEEE Computer Society, 2014.

[2] “Multicore Task Management API (MTAPI) Specification V1.0”. The Multicore Association, 2013.

[3] M. Herlihy, N. Shavit. “On the Nature of Progress”. International Conference on Principles of Distributed Systems (OPODIS), Springer, 2011.

[4] T. Schüle. “Embedded Multicore Building Blocks – Parallel Programming Made Easy“. Embedded World, 2015.

[5] Embedded Multicore Building Blocks – Tutorial, 2015.

[6] Embedded Multicore Building Blocks – Reference Manual, 2015.

Autor

Dr. Tobias Schüle ist Senior Key Expert Engineer, zertifizierter Software-Architekt und Projektleiter bei Siemens Corporate Technology, der zentralen Forschung und Entwicklung von Siemens. Sein Hauptinteresse gilt neben der parallelen Programmierung der Architektur und Entwicklung von eingebetteten Systemen. Er ist Autor zahlreicher Veröffentlichungen und Coautor des Buchs „Multicore-Software“.

E-Mail: tobias.schuele@siemens.com