对称多处理或虚拟化

最大化软件控制架构的价值与能力

IntervalZero

概要

多核技术的持续进步是软件创新的催化剂，这种创新孕育了新一代嵌入式系统。这些系统因为能并列运行多项任务，所以开发成本更低，性能更高，还具有扩展性。

这些新系统的理想架构在先前的一份白皮书”软控制架构：复杂系统硬实时设计的突破”中有所描述。本篇白皮书已假设读者具备基本的了解，对于软件控制架构如何取代 FPGA/DSP/PowerPC，并通过一个微软 Windows 的硬实时插件—专用 RTOS 解决方案—来提供突破性的成本、效能和可扩展效益。

系统开发人员已开始利用软件控制架构在多样化的市场像是工业自动化、医疗系统、检测，以及数字媒体中获得持续的竞争优势。

本篇白皮书探讨了两种普遍且竞相提供软件控制架构价值的多处理方法。分别是对称多处理（SMP）和非对称多处理（ASMP），后者常被称为虚拟化或虚拟机管理程序（hypervisor）技术。

这两种方法各有其特点和挑战，但对于那些试图最大化扩展性并通过充分利用 x86 多核的力量（4 核、6 核、8 核及以上）来最小化延迟的开发者来说，SMP 显然更具优势。

我们将会看到，虚拟化/ASMP 确实在初期成本降低方面提供了短期成效，但对于需要扩展性、精准度和效能，且要求最低延迟的真正实时系统来说，却是一种死胡同的技术。

目前虚拟化/ASMP 虽然在单芯片上是主流技术，但这种镜像式的现状极为局限，且缺乏灵活性、抑制效能，当开发人员想跨越双内核时也阻碍了扩展性。

由于支持 SMP 的架构是动态的—与虚拟化/ASMP 的静态架构不同—因此能为开发人员提供更广泛的选择以简化和精简开发流程，同时充分利用多核处理能力，开发出能够改变竞争格局的系统。

这对于那些开发具有复杂人机界面（HMI）和严苛硬实时与控制要求的嵌入式系统开发人员来说尤其如此。大型医疗系统如核磁共振（MRI）和数字媒体混音控台，就是具有这些需求的例子。

如上所述，对于软件控制架构的详细描述可参考早期的白皮书，但为了提供相关背景资料给本篇文章的比较信息，我们在此涵盖了该白皮书的重点概要。

长期以来，FPGA 和 DSP 在运动控制和其他复杂、高精确度和高效能系统的硬实时市场上占有主导地位。但现在情况已经不同了，随着科技进步，OEM 厂家能够部署软件控制架构，从而取代大部分的专用硬件。

长期以来，FPGA 和 DSP 在运动控制和其他复杂、高精确度和高效能系统的硬实时市场上占有主导地位。但现在情况已经不同了。

有利于采用支持 SMP 软件控制架构的主要趋势包括:

- 日益强大的 x86 处理器技术;
- 朝向更多商用现成（COTS）硬件和软件的发展趋势;
- 以太网总线的进步及可用性;
- 系统设计中组件的融合;以及
- 以触控为中心的可用性—特别是多点触控—和运动感测技术。

利用这些趋势所形成的软件控制架构采用了多核 x86，可以在单个多核的芯片上同时运行 Windows（包括带有触控和手势技术的 Windows 7）和支持对称多处理的硬实时插件例如英特蒙的 RTX64。

打造软件控制架构可使用不同的技术，但为了实现最大价值，开发人员必须牢记八个关键的成功特征:

1. 通用的集成开发环境（IDE）和世界级的图形用户界面（GUI）— Microsoft Visual Studio 和 Windows 操作系统，包括 Windows 10 和 Windows Embedded Standard 7;
2. 直接在多个指定处理器（不是多个个体）执行能支持 SMP 的实时子系统;
3. 所有实时进程都能监看硬件状态;
4. 能够在多个处理器上调度实时线程，或将某些逻辑专用于特定核心并利用 hook API 达到负载平衡;
5. 直接存取共享数据/内存，无需额外副本和进程间通信;
6. 硬件需求最小化—处理器、内存、功率和占用空间;
7. 能够跨内核进行调试;
8. 一次编码和自动扩展的能力。

在 SMP 能够完全满足这八项成功要素的同时，虚拟化/ASMP 只能部分实现一个特性—通过将原本需要两台 PC 的系统集成为单一 PC 以最小化硬件需求（第 6 点）。

虚拟化/ASMP 确实让许多不同的操作系统/应用程序存在于同一个多核硬件上，并共享周边资源如 I/O、串口、以太网、USB 等，也因此似乎是一种降低系统成本和复杂度的好方法。

过去在不同 PC 系统上运行的应用程序，现在已经能集成到单一 PC 中的一块硅芯片上，这样的方式能降低物料成本、保留现有编码、也不需要重写和重新设计架构。然而，这正是事情陷入死胡同的地方，虚拟化/ASMP 架构的静态本质限制了自身的扩展性，因此无法成为可行的长期解决方案。

例如，在虚拟化/ASMP 中，多处理器配置需要的所有元素—独立的实时操作系统（RTOS）、独立的代码库、独立的开发工具和独立的开发团队—都会镜像到单一系

统上，客户管理的个体并没有减少，反而是增加了一个用来管理的 hypervisor。

简而言之，虽然虚拟化/ASMP 在硬件成本降低上有所收获，但并没有相应的软件突破—像 SMP 那样—能让开发人员在多核上垂直扩展并打造在市场上具备差异化的系统。

例如，随着嵌入式开发人员追求利用越来越多的核心，通过将较大的功能分散到不同核心以提高效能；或将核心分散为相关功能；或指派核心进行特定的计算和数学功能—这些都能通过 SMP 达成—而虚拟化/ASMP 的镜像架构则会带来重大挑战。

虚拟化/ ASMP 目前虽然在单芯片上是主流技术，但这种镜像式的现状极为局限，且缺乏灵活性、抑制效能，当开发人员跨越双核时也阻碍了可扩展性。

让我们再深入探讨其他七种成功特征，就能看出 SMP 的相对优势，和虚拟化/AMSP 在提供硬实时软件控制架构方面的弱点。

1. 通用的集成开发环境（IDE）和世界级的图形用户界面（GUI）— Microsoft Visual Studio 和 Windows 操作系统，包括 Windows 10 和 Windows Embedded Standard 7

虽然虚拟化/ASMP 技术可能提供，也可能不提供一个通用的开发环境，但虚拟化并不会搭配一个集成环境。每个操作系统的个体都必须有自己的代码，并对在其他内核上运行的隔离环境中的代码一无所知。虚拟化的缺乏集成性会导致管理成本急剧上升。

2. 直接在多个指定处理器（不是多个个体）执行能支持 SMP 的实时子系统

由支持 SMP 的实时插件（如英特蒙的 RTX64）提供的辅助调度器，能够清楚地检视所有处理器，这确保了内核之间的正确使用和同步。

SMP 架构通过拥有另一个辅助调度器—除了 Windows 调度器之外独立的调度器—而获得很大程度上的好处，并专用于处理所有客户端应用程序的内核。

由支持 SMP 的实时插件（如英特蒙的 RTX64）提供的辅助调度器，能够清楚地检视所有处理器，这确保了内核之间的正确使用和同步。

这能让用户实行一个监督进程，用于监控各个内核上的活动。根据环境需求或用户指令，调度器可使用各种 API 呼叫来分配或集成内核之间的功能，这对于优化吞吐量和通过闲置核心最小化耗电量来说非常有用。

作为范例，我们来看看一个四核的 x86 设备，其中一个内核分配给 Windows，剩下的三个分配给 SMP 子系统。在负载较重的情况下，所有分配给 SMP 的三个内核可能都会 100% 满载，这是硬件的最佳使用方式。然而，如果 100% 的负载只在 20% 的工作周期内发生呢？这样就变成多核架构的低效使用，会浪费电力和效能。

SMP API能让用户编写监控进程或负载平衡器，将功能集成到单一核心上，使得系统能闲置/停用两个内核以减少电力消耗。

相同的 API 呼叫也能在单个内核超载的情况下，将线程/进程分配到利用率较低的内核，从而提高处理能力和吞吐量。

虚拟化/ASMP 架构在执行管理各内核之间的进程监督功能或负载平衡方面则存在着很大的困难，因为每个内核都有单独的应用程序映像和实时操作系统。每个应用程序只了解自己内部的调度并且是通过基于硬件设计的进程间通信，这意味着为了在不同内核之间正确移动和调度线程/进程，原始应用程序必须预先建置所有必要的通信和支持功能以达到负载平衡，但这通常不在单核设备（如 DSP）的设计规范中。而这些负载平衡 API 对 SMP 调度器来说即是原生的 API。

此外，虚拟化/ASMP 架构承袭了传统多处理器设计的限制，例如复制的设计可能在 CPU、内存、I/O 等方面存在一些效能限制，这些限制来自于该设计使用的 DSP。

这些限制不一定会被带到新的多核设备，但由于应用程序是直接导入到虚拟化/ASMP 设计中，因此这些限制仍会存在，结果就是：设计不够理想。

3. 所有实时进程都能监看硬件状态

在 SMP 架构中，客户端应用程序直接运行在内核上，并能无障碍地存取 I/O，简而言之，所有实时进程都能看到硬件资源的状态。在虚拟化/ASMP 架构中则不是这样。

在 SMP 架构中，客户端应用程序直接运行在内核上，并能无障碍地存取 I/O，简而言之，所有实时进程都能看到硬件资源的状态。

由于虚拟化/ASMP 架构只是直接导入曾在专用设备上运行的独立应用程序，因此并没有考虑到主机和客户端应用程序要如何充分利用新设备的功能和特性，这种缺乏协调的状况最终会导致大量的硬件竞争。

让我们来看看将一个Windows设备和两个DSP集成到单个四核心x86的例子：每个不同的应用程序都有其专属的硬件/外围设备，包括从串口、以太网控制器，再到 USB 埠，这里就是虚拟化/ASMP 架构需要一个 hypervisor 或抽象层来管理和仲裁所有不同应用程序之间外围设备的地方，而这对不同的应用程序产生了一种独占特定硬件的错觉。尽管可行，但这种仲裁会导致系统延迟，随着应用程序数量的增加，竞争越多，延迟也会越大。

SMP 拥有在内核之间移动线程/进程所需的 hook API，这是负载平衡和正确使用多核架构的必要基石。负载平衡是确保最佳吞吐量和内核利用率的关键。

让我们再次看看将一个Windows设备和两个DSP集成到单个四核 x86 的范例—这次使用支持 SMP 的 Windows 实时插件。重新架构SMP的第一步就是将所有时间关键或CPU密集型的线程/进程安排到辅助SMP调度器，同时保留 Windows 调度器的功能。接着，设计师确定外围设备/驱动程序的位置：非时间关键的Windows子系统或SMP实时子系统，一旦驱动程序被指派，该特定子系统就会对该外围设备拥有不受阻碍的独占权限。这种架构非常具有扩展性，因为随着内核数量的增加，SMP 调度器就能通过简单的同步机制（号志、互斥锁等）轻松控制任何内核上的线程/进程和驱动程序。

在某些情况下驱动程序还能共享，而在这些情况下，子系统之间可以导出驱动程序API以共享外围设备和同步呼叫。这说明 SMP 不仅提供了极大的效能，并且在打造可扩展和并行性的系统方面也提供了高度的灵活性。随着 Intel 和 AMD 多核设备的快速发展，扩展性和并行性变得越来越重要。

4. 能够在多个处理器上调度实时线程，或将某些逻辑专用于特定核心并利用 hook API 达到负载平衡

SMP拥有在内核之间移动线程/进程所需的hook API，这是负载平衡和正确使用多核心架构的必要基石。

负载平衡是确保最佳吞吐量和核心利用率的关键。

如上所述，虚拟化/ ASMP 在负载平衡方面有困难，因为原始系统分割是沿用于复制的旧有设计，如果没有额外的副本和在不同内核之间进行十分复杂的协调，就无法轻松地在处理器之间移动需要的功能。

5. 直接存取共享数据/内存，无需额外副本和进程间通信

因为 SMP 架构只是和主机应用程序共享内存，所以在不同内核/应用程序之间共享数据是很自然的事，这也消除了任何数据复制的要求，因此能减少内存需求和提高效能。

虚拟化/ASMP 架构则必须对数据和程序代码进行多次复制，因为原始设计的限制被承袭下来—例如进程间通信以及程序和数据对独占内存的要求，而冗余数据的缓冲就会增加延迟。

6. 硬件需求最小化—处理器、内存、功率和占用空间

SMP 架构和虚拟化/ ASMP 都能集成硬件需求，减少25-50%的运算硬件成本。然而，虚拟化/ASMP并没有获得满分，因为硬件消耗并未优化，而且在应用层级看不到的 hypervisor 层级可能会出现竞争和延迟。

相比之下，SMP 架构因为在内核之间以及两个子系统(Windows 和 SMP) 之间共享大量数据，而拥有更高的效率和更小的占用空间，这减少了所需要的内存量，并进一步降低系统成本。此外，藉由在不同核心间组织线程/进程所达成的更高效率和性能，往往能减少对处理器的需求，进而降低系统成本。

7. 能够跨内核进行调试

这是非常重要的。

在虚拟化/ ASMP 架构中，竞争和延迟可能会发生在 hypervisor 层级，但在应用层级却无法察觉，这尤其麻烦—使得调试极其困难，并且使系统故障成为主要隐忧。

为了适当地调试多核系统，一套完整的工具必须要能识别所有操作系统并且观察的到整个系统。在虚拟化/ ASMP 架构中，因为原始应用程序和实时操作系统 (RTOS) 是直接导入的，IDE 就无法识别运行在其他内核的客户端应用程序。

在这种情况下，需要使用不同的工具来调试不同内核或应用程序间的问题，这在双核系统并不是特别困难，但随着系统扩展到四核及以上的时候，就会变得越来越令人难以应付。

想象一下在调试四核或六核系统时的复杂性，其中有更多四个或六个独立的IDE尝试调试不同的内核，尽管各个 IDE/调试器本身很强大，但对其他环境的可见性却非常有限。当处理 hypervisor 为了仲裁曾经专属于不同应用程序的共享外围设备而造成的额外延迟时，这种有限的调试可见性问题就会变得更加严重。

在处理时间关键型应用程序时也值得注意的是，hypervisor 造成的延迟会产生一个在应用层级必须考虑进去的未知时间常数。

最后，除了调试的复杂性以外，虚拟化/ASMP 架构通常要求用户维护所有不同的工具和操作系统。随着内核数量的增加，维护这些不同的工具会增加成本并带来额外的支持负担。

在虚拟化/ ASMP 架构中，竞争和延迟可能会发生在 hypervisor 层级，但在应用层级却无法察觉，这尤其麻烦—使得调试极其困难，并且使系统故障成为主要隐忧。

SMP 架构在开发工具和调试方面采用了更简化的方法。

SMP 应用程序建置在 Visual Studio IDE 中，并且能完全察觉 Windows 和 SMP 调度器。SMP 子系统使用一个辅助调度器来管理实时子系统中的所有内核，由于 SMP 子系统只有一个调度器，因此随着内核/应用程序数量的增加，调试的复杂度并不会跟着变大。Visual Studio 的 RTX 调试工具可以完全控制和查看所有内核以及两个调度器—Windows 和 SMP。

在 SMP 架构中的应用程序能够直接与其外围设备互动并拥有控制权，从而消除了应用程序必须考虑的任何额外延迟。这种紧密的集成使开发和调试变得简单明了，用户可以在 Visual Studio 中轻松地设置 Windows 和 SMP 子系统之间的断点。SMP 使用单一的 IDE 大大简化和精简了调试和工具维护，并且非常具有成本效益。

由于 SMP 子系统只有一个调度器，因此随着内核/应用程序数量的增加，调试的复杂度并不会跟着变大。

8. 一次编码和自动扩展的能力

这点在上面已提及过，但由于并行处理是非常重要的，因此值得更详细的解释。

SMP 架构包括可以指派处理器亲缘性和理想处理器的 API，这些 API 让开发人员只需要为理想的多核架构编写一次程序，软件就能随着更多可用内核的增加而自动扩展到设备上。

相反的，从不同的多处理器转换成单一芯片之后，在虚拟化/ ASMP 的环境中为各种系统设计适当利用不同内核倍数会变得困难。

如果没有大幅度的重写程序代码，虚拟化/ ASMP 设计必须在每个新增的内核上放置原始客户端应用程序的多个副本/映像，但这样就无法利用原本能藉由组合那些在特定内核上运行而获益的线程/进程来提高效能和吞吐量，变成很多余且浪费资源。

开发人员非常有理由担忧虚拟化/ASMP 无法减轻传统多处理器系统中存在的复杂性和低效率—多种工具环境、多个开发团队和多个操作系统。

而从不同的芯片集成到外围设备共享的多核架构时，资源竞争可能会成为各个开发团队面临的重大挑战。

以前专用于每个应用程序的外围设备现在变成共享，不可避免地意味着当需要的外围设备正在忙碌时，有些操作就必须等待，这不仅效率低下，还和从多核硬件中获得最大处理速度和性能的目标背道而驰。简而言之，闲置的芯片是资源的浪费，开发人员要真正降低系统成本，必须时时刻刻将所有处理的潜力最大化。

如前所述，这些挑战在 SMP 架构中根本不存在。

简单来说，尽管虚拟化/ ASMP 可能适用于单一任务或较不复杂的环境—这也是桌面虚拟化容易被接受的原因—但并不适合用来解决必须具备硬实时、精确性、灵活性、加速上市时间和扩展性的复杂多任务系统。

事实上，要让虚拟化/ASMP 成为复杂嵌入式系统的可靠选项，就需要一个 Windows hypervisor 扩展程序，也就是外挂一个实时 SMP 产品并扩展内核和外围设备的存取。

有了 SMP，开发人员在重新思考和设计系统的时候会有很多选择，不仅能充分利用多核能力，还可以改善开发流程，实现长期经济效益和扩展性。

通过使用支持SMP的硬实时子系统为Windows/x86系统增加第二个独立的调度器，系统开发人员获得了一个适合多处理能力最大化的软件控制架构，同时提供超越系统集成的显著经济和性能效益。

结论

充其量而言，虚拟化/ASMP 代表了从多个系统到多核的短期、低价值集成路径。然而，对于一个真正简化和精简、高性能、可扩展、高效且具有长期价值的架构来说，支持 SMP 的软件控制架构还是最佳选择。

对称多处理（SMP）vs. 虚拟化（ASMP）

Windows/X86 实时嵌入式解决方案的 8 个关键设计元素	优势	SMP	ASMP
1. 通用的集成开发环境（IDE）和世界级的图形用户界面（GUI）— Microsoft Visual Studio 和 Windows 操作系统，包括 Windows 10 和 Windows Embedded Standard 7	集成 & 生产力	●	○
2. 直接在多个指定处理器（不是多个个体）执行能支持 SMP 的实时子系统	效能 & 可维护性	●	○
3. 所有实时进程都能监看硬件状态	控制	●	○
4. 能够在多个处理器上调度实时线程，或将某些逻辑专用于特定核心并利用 hook API 达到负载平衡	控制 & 效能	●	○
5. 直接存取共享数据/内存，无需额外副本和进程间通信	质量 & 效能	●	○
6. 硬件需求最小化—处理器、内存、功率和占用空间	效能	●	◐
7. 能够跨内核进行调试	效能	●	○
8. 一次编码和自动扩展的能力	可扩展性	●	○