HPE COMPILER GPU OFFLOADING

Steve Abbott
HPE Cray Programming Environment & CORAL-2 Centers of Excellence
April 29, 2022
SOME RELEVANT PREVIOUS TALKS WITH SOME USER EXPERIENCE
Ordered chronologically

- “Experiences in Implementing OpenMP offload support in Fortran” by Kostas Makrides (HPE) & Aaron Black (LLNL).
- “Asynchronous 3-D FFTs using OpenMP offload for extreme problem sizes” by Kiran Ravikumar (Georgia Tech) et al.
- “Using OpenMP to Harness GPUs for Core-Collapse Supernova Simulations with GenASiS” by Reuben Budiardja (ORNL).
- “OpenMP experiences with Thornado” by Austin Harris (ORNL).
OUTLINE

• Broader HPE Cray PE accelerator support
• General compiler overview
• Offloading models
• Offloading feature highlights and best practices
Provides compiler and library choice, performance, and programmability

- Multiple programming environments
- Compiler interoperability
- Automatically uses our math, scientific, and communication libraries with chosen compiler
- Can use debug and profiling tools with chosen compiler

<table>
<thead>
<tr>
<th>HPE Cray Programming Environment</th>
<th>AMD Programming Environment</th>
<th>Intel Programming Environment</th>
<th>NVIDIA Programming Environment</th>
<th>GNU Programming Environment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Compiling Environment (CCE)</td>
<td>AMD AOCC and ROCm compilers</td>
<td>Intel® C, C++, and Fortran compilers</td>
<td>NVIDIA compilers</td>
<td>GNU Compiler Collection</td>
</tr>
</tbody>
</table>

- Cray MPI and SHMEM
- Performance Analysis Tools
- Debugger Support Tools
- Scientific and Math Libraries
- Environment Setup and Compiling Support
- Third Party Products
Comprehensive set of tools for developing, porting, debugging, and tuning of HPC applications on HPE & HPE Cray systems

**HPE CRAY PROGRAMMING ENVIRONMENT**

### Development

**Programming Models**
- HPE Cray MPI
- SHMEM
- OpenMP | OpenACC
- AMD ROCm HIP | NVIDIA CUDA
- UPC | Fortran co-arrays

**Chapel**

**Programming Environments**
- Compiling Environment
  - GNU
  - Intel Programming Environment
  - AMD Programming Environment
  - NVIDIA HPC SDK

**Optimized Libraries**
- LibSci (BLAS)
- LAPACK & ScalAPACK
- LibSci_ACC
- IRT
- FFTW

**I/O Libraries**
- NetCDF
- HDF5

**Deep Learning Plug-in**

**Development Tools**
- C
- C++
- Fortran
- Python
- Chapel
- R

**Debugging**

**Comparative Debugger**
Compare two versions of an application

**GDB for HPC**
Parallelized gdb for HPC

**Valgrind for HPC**
Memory debugging at scale

**Stack Trace Analysis Tool**
Stack tracing at scale

**Tool for Abnormal Termination Processing**
Manage core files at scale

### Performance Analysis & Optimization

**Performance Analysis Tool (PAT)**
Whole program performance analysis, exposing wide set of indicators, identifying bottlenecks and automatically generating suggestions to improve performance.

**Visualization Tool**
Complements text reports with summary of performance data in graphs and charts, allowing users to drill down and resolve issues.

**Code Parallelization Assistant**
Reveal hidden potential of an application via code restructuring

### Setup & Runtime

**Environment Setup**
- Tool Enablement (for Spack, CMake, EasyBuild, etc)
- Modules / Lmod

**Supported systems:**
- HPE Cray EX supercomputers
- HPE Apollo 2000
- HPE Apollo 6500
- HPE ProLiant DL systems
- Legacy Cray systems
COMPILING ENVIRONMENT ADVANTAGES

Performance and programmability

- Classic Cray Fortran compiler
  - Fortran 2018 (w. co-arrays)
  - Proprietary front end, optimizer; HPE-modified LLVM backend
  - C and C++ compiler
    - C11 and C++17; UPC
    - HPE-modified closed-source build of Clang+LLVM compiler

Offloading support
- OpenMP 5.0 and partial 5.1
- OpenACC 2.7 – focus on Fortran, 3.0 in 2022
- HIP – AMD GPUs only

Fully integrated heterogeneous optimization capability

- Providing consistency across all HPE and Cray HPC systems
  - NVIDIA GPUs – Cray XC and CS systems today, Cray EX in 2022
  - AMD GPUs – HPE Cray EX and Apollo systems

- Supporting:
  - x86-64 (both Intel and AMD) processors
  - ARM-based processors
  - NVIDIA accelerators
  - AMD accelerators

Integration with Program Development Tools

- Performance analysis tools exploit compiler's whole program analysis
- Interfaces through CCE's Program Library technology, an application-wide repository
- Code Parallelization Assistant leverages compiler analyses
- Compiler optimization feedback also supplied by performance reports for application tuning

Focus on application portability and investment protection

- Focus on compliance and language support:
  - Languages: Fortran, C/C++
  - Programming models: OpenMP, OpenACC, and PGAS
  - Encourage coding safety with strict standards compliance
  - Support current versions of specifications

Integration with Program Development Tools

- Providing consistency across all HPE and Cray HPC systems
  - NVIDIA GPUs – Cray XC and CS systems today, Cray EX in 2022
  - AMD GPUs – HPE Cray EX and Apollo systems

- Supporting:
  - x86-64 (both Intel and AMD) processors
  - ARM-based processors
  - NVIDIA accelerators
  - AMD accelerators
GENERAL COMPILER OVERVIEW
HPE CRAY COMPILING ENVIRONMENT (CCE)

- A major part of the broader HPE Cray Programming Environment (CPE) supported on HPE systems
  - Compilers + Math & Communication Libraries + Debuggers + Performance Analysis Tools
- Fortran compiler
  - Proprietary front end and optimizer; HPE-modified LLVM backend
  - Fortran 2018 support (including coarray teams)
- C and C++ compiler
  - HPE-modified closed-source build of Clang+LLVM compiler
  - C11 and C++17 support
  - UPC support
- Offloading support
  - NVIDIA GPUs
  - AMD GPUs
  - OpenMP 4.5 and near-complete 5.0
  - OpenACC 2.0 – Fortran only
  - HIP – AMD GPUs only
CCE COMPILER RELEASE AND VERSIONING

- Two major releases a year (~Q2 and ~Q4)
  - CCE codebase and version based off latest Clang major release (lag by ~2 months)
- Monthly minor updates in between
  - Continue for 4 months after each major release
- Examples
  - CCE 12.0 – based on Clang 12.0 – Jun 2021
  - CCE 13.0 – based on Clang 13.0 – Nov 2021
  - CCE 14.0 – based on Clang 14.0 – May 2022
  - CCE 15.0 – based on Clang 15.0 – Nov 2022 (tentative)
- Release cadence and versioning changed in CCE 10.0
  - Older versions of CCE do not correspond to Clang/LLVM version numbers
CCE COMPILER DOCUMENTATION

• Man pages of interest
  • cc, CC, ftn – CCE compiler driver documentation
  • craycc, crayCC, crayftn – CCE C, C++, and Fortran compiler documentation
  • intro_openmp – CCE OpenMP documentation
  • intro_openacc – CCE OpenACC documentation
  • intro_directives – CCE compiler directives

• PDF manuals
  • Search at: https://support.hpe.com/hpesc/public/home
    – S-2179 for the release overview
    – S-3901 for the Fortran reference manual
    – S-5212 for the C/C++ quick reference guide

• Release information
  • module help cce/X.y.z
CCE OFFLOADING MODELS
CCE OPENMP SUPPORT

- Uses proprietary OpenMP runtime libraries
- Supports cross-language and cross-vendor OpenMP interoperability
- Implements HPE-optimized code generation for OpenMP offload regions
- Supports asynchronous "nowait" GPU operations with "depend" clauses
- Supports OpenMP allocators (e.g., CPU “pinned”, GPU “shared” and “managed”)
- Full OpenMP 4.5 support for Fortran, C, and C++
- OpenMP 5.x – in progress, implementation phased in over several CCE releases
  - See release notes and intro_openmp man page for full list of supported features
  - OpenMP 5.0 is near complete as of CCE 14.0 (May 2022)
  - OpenMP 5.1/5.2 support in progress for 2022-2023
## CCE OPENMP 5.0 STATUS

<table>
<thead>
<tr>
<th>CCE 10.0 (May 2020)</th>
<th>CCE 11.0 (Nov 2020)</th>
<th>CCE 12.0 (Jun 2021)</th>
<th>CCE 14.0 (May 2022)</th>
</tr>
</thead>
<tbody>
<tr>
<td>• OMP_TARGET_OFFLOAD</td>
<td>• noncontig update</td>
<td>• device_type (Fortran)</td>
<td>• task reduction (Fortran)</td>
</tr>
<tr>
<td>• reverse offload</td>
<td>• map Fortran DVs</td>
<td>• affinity clause</td>
<td>• task modifier (Fortran)</td>
</tr>
<tr>
<td>• implicit declare target</td>
<td>• host teams</td>
<td>• conditional lastprivate (C/C++)</td>
<td>• target task reduction (Fortran)</td>
</tr>
<tr>
<td>• omp_get_device_num</td>
<td>• use_device_addr</td>
<td>• simd if (C/C++)</td>
<td>• simd (Fortran)</td>
</tr>
<tr>
<td>• OMP_DISPLAY_AFFINITY</td>
<td>• nested declare target</td>
<td>• iterator in depend (C/C++)</td>
<td></td>
</tr>
<tr>
<td>• OMP_AFFINITY_FORMAT</td>
<td>• allocate directive</td>
<td>• depobj for depend (C/C++)</td>
<td></td>
</tr>
<tr>
<td>• set/get affinity display</td>
<td>• allocate clause</td>
<td>• task reduction (C/C++)</td>
<td></td>
</tr>
<tr>
<td>• display/capture affinity</td>
<td>• order(concurrent)</td>
<td>• task modifier (C/C++)</td>
<td></td>
</tr>
<tr>
<td>• requires</td>
<td>• atomic hints</td>
<td>• simd nontemporal (C/C++)</td>
<td></td>
</tr>
<tr>
<td>• unified_address</td>
<td>• default nonmonotonic</td>
<td>• scan (C/C++)</td>
<td></td>
</tr>
<tr>
<td>• unified_shared_memory</td>
<td>• imperfect loop collapse</td>
<td>• lvalue list items for depend</td>
<td></td>
</tr>
<tr>
<td>• atomic_default_mem_order</td>
<td>• pause resources</td>
<td>• mutexinoutset (C/C++)</td>
<td></td>
</tr>
<tr>
<td>• dynamic allocators</td>
<td>• atoms in simd</td>
<td>• taskloop cancellation (C/C++)</td>
<td></td>
</tr>
<tr>
<td>• reverse_offload</td>
<td>• simd in simd</td>
<td>•</td>
<td></td>
</tr>
<tr>
<td>• combined master constructs</td>
<td>• detachable tasks</td>
<td>•</td>
<td></td>
</tr>
<tr>
<td>• acq/ref memory ordering (Fortran)</td>
<td>• omp_control_tool</td>
<td>•</td>
<td></td>
</tr>
<tr>
<td>• deprecate nested-var</td>
<td>• OMPT</td>
<td>•</td>
<td></td>
</tr>
<tr>
<td>• taskwait depend</td>
<td>• OMPD</td>
<td>•</td>
<td></td>
</tr>
<tr>
<td>• simd nontemporal (Fortran)</td>
<td>• declare variant (Fortran)</td>
<td>•</td>
<td></td>
</tr>
<tr>
<td>• lvalue map/motion list items</td>
<td>• loop construct (C/C++)</td>
<td>•</td>
<td></td>
</tr>
<tr>
<td>• allow != in canonical loop</td>
<td>• metadirectives (C/C++)</td>
<td>•</td>
<td></td>
</tr>
<tr>
<td>• close modifier (C/C++)</td>
<td>• mapper (C/C++)</td>
<td>•</td>
<td></td>
</tr>
<tr>
<td>• extend defaultmap (C/C++)</td>
<td>• extend defaultmap (Fortran)</td>
<td>•</td>
<td></td>
</tr>
</tbody>
</table>

### Future CCE Release

| · loop construct (C/C++) | · mapper (Fortran) |
| · iterator in depend (Fortran) | · non-rectangular loop collapse (Fortran) |
| · depobj for depend (Fortran) | · uses_allocators |
| · concurrent maps | · taskloop cancellation (Fortran) |
| · scan (Fortran) | · target task reduction (C/C++) |

Refer to CCE release notes or intro_openmp man page for current implementation status
OPENMP INTEROPERABILITY

- OpenMP CPU interoperability
  - CCE’s libcraymp behaves as drop-in replacement for Clang’s libomp and GNU’s libgomp
  - GNU OpenMP interface support is currently limited to OpenMP 3.1 constructs
- OpenMP GPU interoperability
  - CCE’s libcrayacc behaves as drop-in replacement for Clang’s libomptarget
  - No planned support for GNU OpenMP offload interface
  - Device code relies on each vendor’s device runtime library
  - Each vendor’s device code is linked into a separate “device image”
  - CCE OpenMP offload linker tool handles device unbundling and linking
  - Requires linking with CCE, or manually invoking the CCE OpenMP offload linker tool
CCE supports OpenACC 2.0+ for Fortran
C/C++ OpenACC support was dropped in CCE 10.0
Full OpenACC 3.2 support planned for a future CCE release
CCE OpenMP and OpenACC implementations share a common codebase
  • Significant overlap in both compiler and runtime library
  • Same performance should be achievable with either model
**CCE OPENMP/OPENACC FLAGS**

<table>
<thead>
<tr>
<th>Capability</th>
<th>CCE Fortran Flags</th>
<th>CCE C/C++ Flags</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enable/Disable OpenMP (disabled at default)</td>
<td>-f[no-]openmp -h[no]omp</td>
<td>-f[no-]openmp</td>
</tr>
<tr>
<td>Enable/Disable OpenACC (enabled at default)</td>
<td>-h[no]acc</td>
<td>N/A</td>
</tr>
<tr>
<td>Enable HIP</td>
<td>N/A</td>
<td>-x hip --rocm-path=$ROCM_PATH -L $ROCM_PATH/lib --lamdhip64</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Offloading Target</th>
<th>All CCE Compilers (accel modules)</th>
<th>CCE C/C++ (optional flags)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Native Host CPU</td>
<td>craype-accel-host</td>
<td>(default without flags; no warning)</td>
</tr>
<tr>
<td>NVIDIA Volta</td>
<td>craype-accel-nvidia70</td>
<td>-fopenmp-targets=nvptx64 -Xopenmp-target -march=sm_70</td>
</tr>
<tr>
<td>AMD MI100</td>
<td>craype-accel-amd-gfx908</td>
<td>-fopenmp-targets=amdgcn-amd-amdhsa -Xopenmp-target=amdgcn-amd-amdhsa -march=gfx908</td>
</tr>
<tr>
<td>AMD MI250X</td>
<td>craype-accel-amd-gfx90a</td>
<td>-fopenmp-targets=amdgcn-amd-amdhsa -Xopenmp-target=amdgcn-amd-amdhsa -march=gfx90a</td>
</tr>
</tbody>
</table>
CCE HIP offloading relies on ROCm headers, host libraries, and device bitcode libraries
CCE OpenMP offloading relies on ROCm host libraries and device bitcode libraries
Device bitcode libraries require a matching LLVM version between CCE and ROCm
CCE OpenMP interoperability relies on compatible Clang OpenMP runtime ABI

<table>
<thead>
<tr>
<th>CCE Version</th>
<th>HIP/OpenMP (CCE Only)</th>
<th>OpenMP Interop (CCE + ROCm)</th>
</tr>
</thead>
<tbody>
<tr>
<td>CCE 13.0.0</td>
<td>ROCm 4.1 – 4.5</td>
<td>ROCm 4.2 – 4.3</td>
</tr>
<tr>
<td>CCE 13.0.1</td>
<td>ROCm 4.1 – 4.5</td>
<td>ROCm 4.2 – 4.5</td>
</tr>
<tr>
<td>CCE 13.0.x</td>
<td>ROCm 4.1 – 4.5</td>
<td>ROCm 4.2 – 4.5</td>
</tr>
<tr>
<td>CCE 14.0.0</td>
<td>ROCm 5.0 – 5.1</td>
<td>ROCm 5.0 – 5.1</td>
</tr>
</tbody>
</table>
CUDA API stability means compatibility is less constrained

CCE runtime uses the CUDA Driver API

Offload regions are compiled to PTX and passed to NVIDIA toolchain for assembly

Testing may be limited to what’s supported by underlying system software releases
CCE HIP SUPPORT

- CCE 11.0 (Nov 2020) introduced support for compiling HIP source files targeting AMD GPUs
- CCE HIP support leverages AMD’s open-source HIP implementation in upstream Clang/LLVM
- CCE relies on HIP header files and runtime libraries from a standard AMD ROCm install
- CCE does not provide a “hipcc” wrapper – invoke the “CC” compiler driver directly

<table>
<thead>
<tr>
<th>CCE HIP Flag</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-x hip</td>
<td>Enables HIP compilation for subsequent input files (avoid on link line or follow with “-x none”)</td>
</tr>
<tr>
<td>--offload-arch=gfx90a</td>
<td>Specifies the MI250X offload target architecture</td>
</tr>
<tr>
<td>--rocm-path=&lt;ROCM_PATH&gt;</td>
<td>Specifies the location of a ROCm install; not required when $ROCM_PATH environment variable is set</td>
</tr>
<tr>
<td>-f[no-]gpu-rdc</td>
<td>Enables (disables) relocatable device code, producing bundled HIP offload object files and allowing cross-file references in HIP device code (default: -fno-gpu-rdc)</td>
</tr>
<tr>
<td>--hip-link</td>
<td>Enables device linking for bundled HIP offload object files; required when compiling with -fgpu-rdc</td>
</tr>
<tr>
<td>-mlllvm -amdgpu-early-inline-all=true</td>
<td>Optimization flags that AMD’s “hipcc” wrapper script provides; may provide additional performance benefit</td>
</tr>
<tr>
<td>-mlllvm -amdgpu-function-calls=false</td>
<td></td>
</tr>
</tbody>
</table>
CCE OFFLOADING FEATURE HIGHLIGHTS AND BEST PRACTICES
## The Multiple Dimensions of GPU Parallelism

<table>
<thead>
<tr>
<th>AMD</th>
<th>NVIDIA</th>
<th>Description</th>
</tr>
</thead>
</table>
| Work group| Threadblock / CTA | • Loosely-coupled, course-grained parallelism  
• **Collective synchronization prohibited**  
• Performs best with massive parallelism  
• Performance scales with more powerful GPUs |
| Wavefront | Warp         | • Fine-grained, independent parallelism  
• NVIDIA warp size is 32 threads  
• AMD wavefront size is 64 work items |
| Work item | Thread       | • Fine-grained, lock-step parallelism  
• Performs best with stride-1 data accesses  
• Performs best with non-divergent control flow |
### OPENACC/OPENMP CONSTRUCT MAPPING TO GPU

<table>
<thead>
<tr>
<th>NVIDIA</th>
<th>AMD</th>
<th>CCE Fortran OpenACC</th>
<th>CCE Fortran OpenMP</th>
<th>CCE C/C++ OpenMP</th>
<th>Clang C/C++ OpenMP</th>
</tr>
</thead>
<tbody>
<tr>
<td>Threadblock</td>
<td>Work group</td>
<td>acc gang</td>
<td>omp teams</td>
<td>omp teams</td>
<td>omp teams</td>
</tr>
<tr>
<td>Warp</td>
<td>Wavefront</td>
<td>acc worker</td>
<td></td>
<td></td>
<td>omp parallel</td>
</tr>
<tr>
<td>Thread</td>
<td>Work item</td>
<td>acc vector</td>
<td>omp simd</td>
<td>omp simd</td>
<td>omp parallel</td>
</tr>
</tbody>
</table>

- **Current best practice:**
  - Use “teams” to express GPU threadblock/work group parallelism
  - Use “parallel for simd” to express GPU thread/work item parallelism
- **Future direction:**
  - Improve CCE support for “parallel” and ”simd” in accelerator regions
  - Upstream Clang is expanding support for “simd” in accelerator regions

Long-term goal: let users express parallelism with any construct they think makes sense, and CCE will map to available hardware parallelism
**RUNTIME OFFLOADING MESSAGES**

- Environment variable CRAY_ACC_DEBUG=[1-3]
- Emits runtime debug messages for offload activity (allocate, free, transfer, kernel launch, etc)

Program main
```fortran
program main
  integer :: aaa(1000)
  aaa = 0
  !$omp target teams distribute map(aaa)
  do i=1,1000
    aaa(i) = 1
  end do

  if ( sum(abs(aaa)) .ne. 1000 ) then
    print *, "FAIL"
    call exit(-1)
  end if
  print *, "PASS"
end program main
```

ACC: Version 4.0 of HIP already initialized, runtime version 3241
ACC: Get Device 0
ACC: Set Thread Context
ACC: Start transfer 1 items from hello_gpu.f90:4
ACC: allocate, copy to acc 'aaa(:)' (4000 bytes)
ACC: End transfer (to acc 4000 bytes, to host 0 bytes)
ACC: Execute kernel main_$ck_L4_1 blocks:8 threads:128 from hello_gpu.f90:4
ACC: Start transfer 1 items from hello_gpu.f90:7
ACC: copy to host, free 'aaa(:)' (4000 bytes)
ACC: End transfer (to acc 0 bytes, to host 4000 bytes)
PASS
ASYNC OFFLOAD CAPABILITIES

- OpenMP offload “nowait” constructs map to independent GPU streams
  - “depend” clauses are handled with necessary stream synchronization
- Task “detach” support introduced in CCE 11.0 (Nov 2020)
- Cross-device dependences are not yet optimized well – overly conservative synchronization
- Multi-threaded use of GPU is optimized as of CCE 13.0 (Nov 2021) – relaxed locking strategy
CCE OPENMP UNIFIED MEMORY DETECTION

- CCE’s default runtime behavior for OpenMP map clauses is to allocate/transfer GPU memory
- Dynamically enable GPU managed memory for OpenMP map clauses
  - Set env var CRAY_ACC_USE_UNIFIED_MEM=1
  - Triggers runtime check to skip explicit allocate/transfer for managed memory
Dynamically enable GPU unified memory for OpenMP map clauses
  • Set env vars `CRAY_ACC_USE_UNIFIED_MEM=1` and `HSA_XNACK=1`
  • Skips explicit allocate/transfer for all system memory
  • Global "declare target" variables will still be allocated separately (compiler statically emits a device copy)

Statically enable GPU unified memory for OpenMP map clauses
  • Compile with "requires unified_shared_memory" directive
  • Set env var `HSA_XNACK=1`
HPE COMPILER SUMMARY

- CCE is one component of the broader HPE Cray Programming Environment
- Consistent development environment across a wide variety of CPU and GPU targets
- Support for the latest base language standards
  - Fortran 2018 support (including coarray teams)
  - C11 and C++17 support
- Support for several on-node parallel/offloading models
  - OpenMP 4.5, working towards 5.2
  - OpenACC 2.0, working towards 3.2
  - HIP
- Please reach out or file bugs if you have questions or encounter issues
THANK YOU

Steve Abbott
stephen.abbott@hpe.com
## CCE OPENMP ALLOCATOR SPECIALIZATION

<table>
<thead>
<tr>
<th>Use Case</th>
<th>Allocator Mechanism</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>“Pinned” CPU memory</td>
<td>Allocator with “pinned” trait set</td>
<td>• Maps to hipMallocHost</td>
</tr>
<tr>
<td>“Shared” GPU memory</td>
<td>omp_cgroup_mem_alloc predefined allocator</td>
<td>• Maps to static allocation in LDS memory</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Must be lexically specified on “allocate” clause on “teams” construct</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Currently supported for Fortran only</td>
</tr>
<tr>
<td>“Managed” memory</td>
<td>cray_omp_get_managed_memory_allocator_handle()</td>
<td>• Maps to hipMallocManaged</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• CCE-specific extension</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Topic of interest for OpenMP committee</td>
</tr>
</tbody>
</table>