

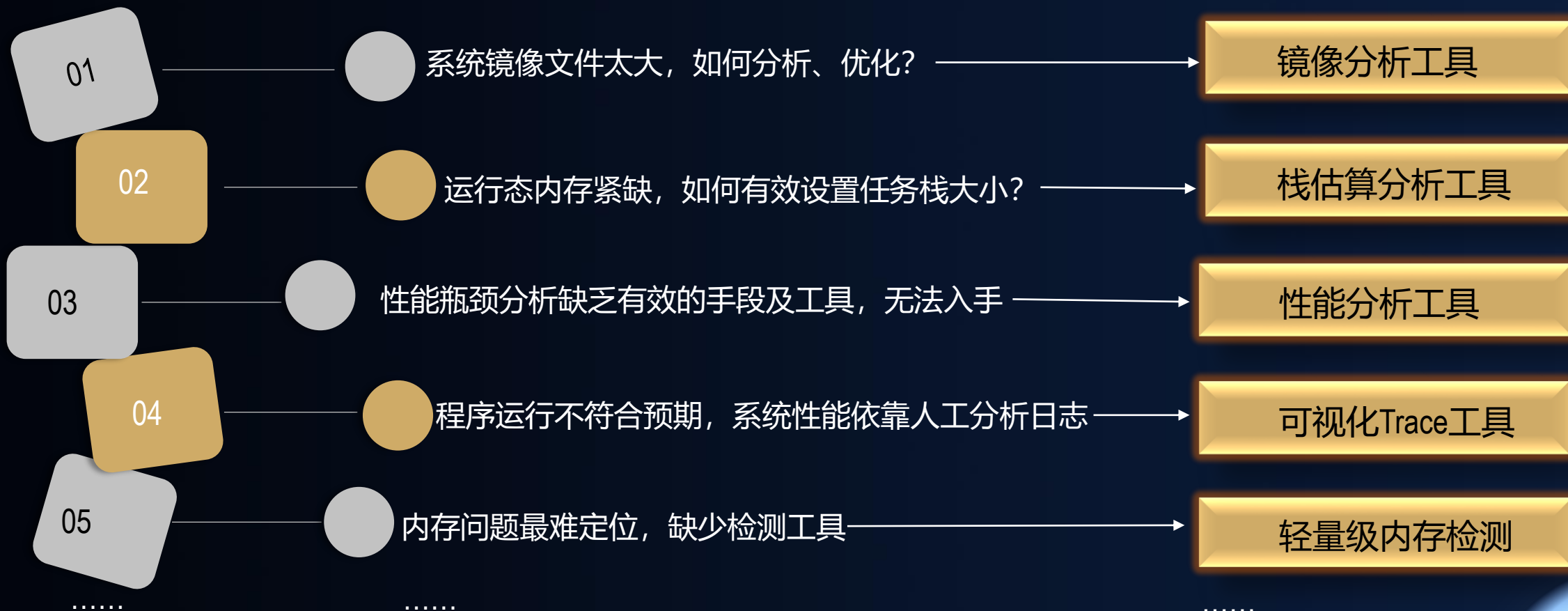
< HDC.Together >

HUAWEI DEVELOPER CONFERENCE 2021

基于LiteOS内核的系统高效调优五大法宝

- **设备开发调优的主要挑战及关键技术**
- **镜像分析工具解读**
- **栈估算分析工具解读**
- **性能分析工具解读**
- **可视化Trace工具解读**
- **轻量级内存检测工具解读**

设备开发调优主要挑战及关键技术



- 设备开发调优的主要挑战及关键技术
- **镜像分析工具解读**
- 栈估算分析工具解读
- 性能分析工具解读
- 可视化Trace工具解读
- 轻量级内存检测工具解读

镜像分析工具解读

开发者典型场景

- 嵌入式设备资源有限，组件增加后，镜像大小增加过多
- 如何可视化地查看系统当中资源占用情况
- 如何快速评估ROM/RAM占用情况
- 如何按文件查看资源占用情况
- 如何按函数模块查看资源占用情况
- 有无工具可以导出程序跳转情况



01 镜像分析，助力镜像文件优化

通过对构建出的 map 文件进行内存占用分析，可解决嵌入式设备开发者难以获取镜像中各模块对ROM/RAM 占用情况的问题。

工具可提供功能

- 支持查看内存区域、详细信息、文件大小及模块大小
- 支持程序符号跳转，支持排序、过滤、导出表格
- 支持按文件/模块查看

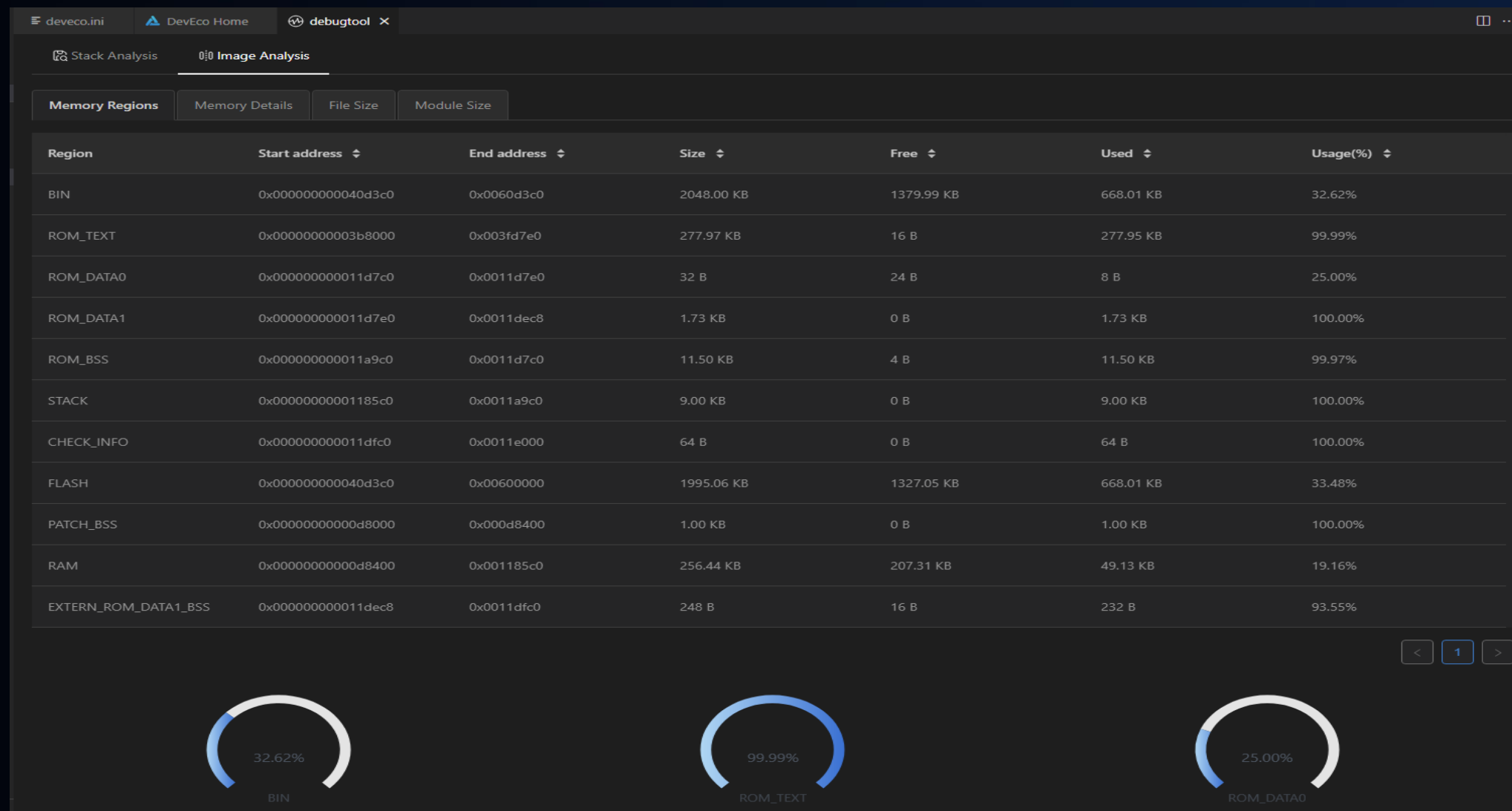
开发者收益

帮助开发者快速评估并优化镜像的ROM/RAM占用情况



镜像分析工具解读

支持查看内存区域总览



镜像分析工具解读

查看内存详细信息

The screenshot shows the DevEco IDE interface with the 'Image Analysis' tool open. The 'Memory Details' tab is selected, displaying a table of memory regions. The table has four columns: Name, Run address(VAM), Load address(LMA), and Size. The regions listed include ROM_DATA0, BIN, FLASH, .text_non_rom, .zInit, .ram_text, .data, .data_rom1, RAM, ROM_DATA1, and PATCH_BSS. The size of each region is shown in bytes or kilobytes.

Name	Run address(VAM)	Load address(LMA)	Size
+ ROM_DATA0			32 B
+ BIN			2048.00 KB
- FLASH			1995.06 KB
+ .text_non_rom	0x0040d3c0	0x0040d3c0	650.44 KB
+ .zInit	0x004afd80	0x004afd80	32 B
+ .ram_text	0x000d8400	0x004afda0	12.78 KB
+ .data	0x000db720	0x004b30c0	3.03 KB
+ .data_rom1	0x0011d7e0	0x004b3ce0	1.73 KB
+ RAM			256.44 KB
+ ROM_DATA1			1.73 KB
+ PATCH_BSS			1.00 KB

镜像分析工具解读

支持按文件查看资源占用情况

deveco.ini DevEco Home debugtool X

Stack Analysis Image Analysis

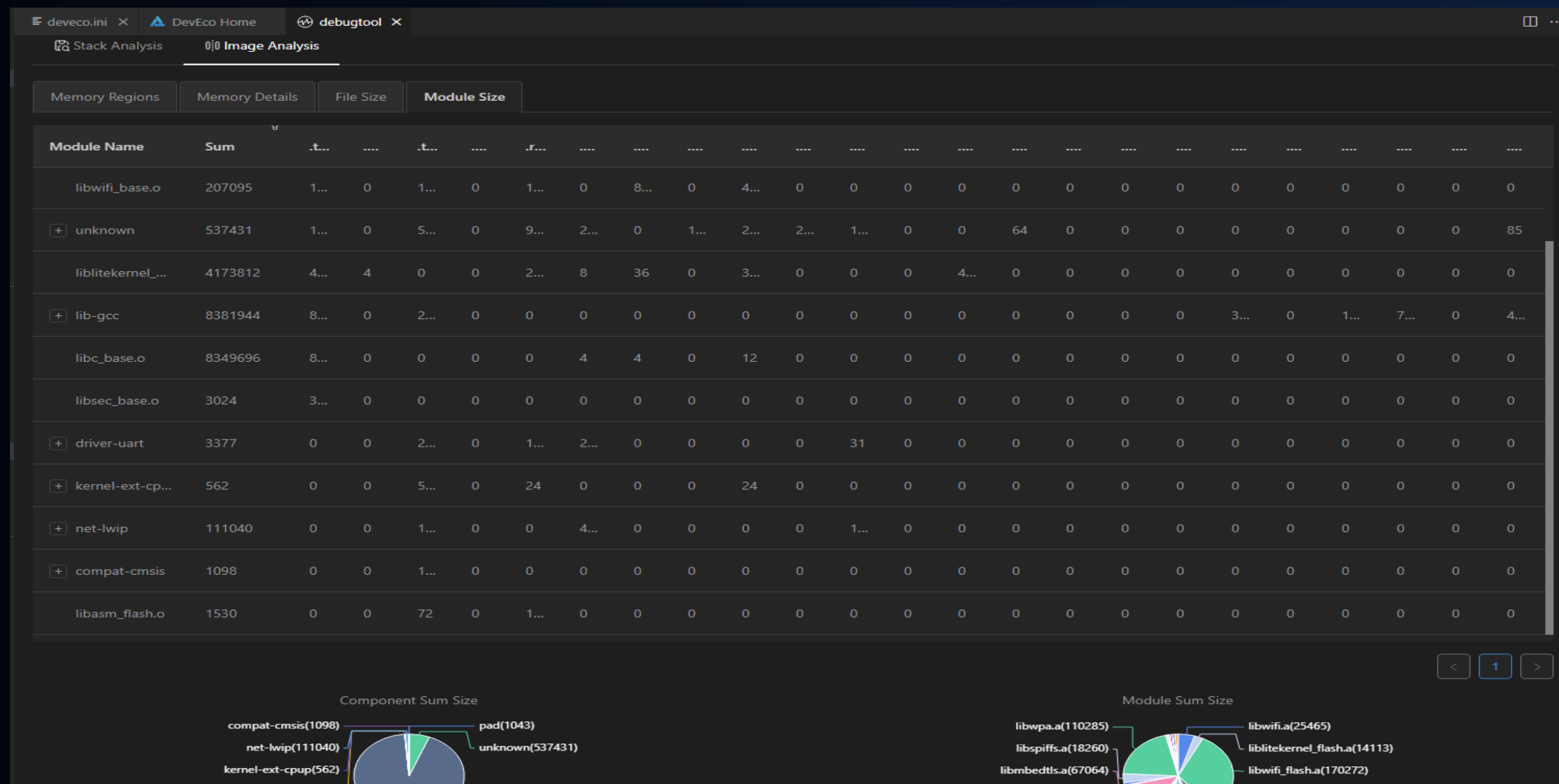
Memory Regions Memory Details **File Size** Module Size

File Name	Sum	.t...t...r...S...	
+ libbsp_base.o	8337457	8...	4	1...	0	0	0	8...	0	3...	0	0	0	0	0	0	0	0	0	0	0	0	0	
+ unknownFile	1043	1...	0	5...	0	28	3...	26	0	86	26	1...	0	9...	0	0	0	0	0	0	0	0	0	
+ libwifi_base.o	207095	1...	0	1...	0	1...	0	8...	0	4...	0	0	0	0	0	0	0	0	0	0	0	0	0	
+ libwifi.a	25465	1...	0	8...	0	0	2	0	0	1...	0	1...	0	0	0	0	0	0	0	0	0	0	0	
+ liblitekernel_...	4173812	4...	4	0	0	2...	8	36	0	3...	0	0	0	4...	0	0	0	0	0	0	0	0	0	
+ libgcc.a	8381944	8...	0	2...	0	0	0	0	0	0	0	0	0	0	0	0	0	0	3...	0	1...	7...	0	4...
+ libc_base.o	8349696	8...	0	0	0	0	4	4	0	12	0	0	0	0	0	0	0	0	0	0	0	0	0	
+ libsec_base.o	3024	3...	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
+ liblitekernel_f...	14113	0	0	1...	0	1...	56	0	1...	0	0	2...	0	0	0	0	0	0	0	0	0	0	0	
+ libwifi_flash.a	170272	0	0	1...	0	2...	1...	0	0	0	0	2...	0	0	0	0	0	0	0	0	0	0	0	
+ libwifi_iiot_app.a	1445	0	0	1...	0	0	23	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	

< 1 2 3 4 >

镜像分析工具解读

支持按模块查看资源占用情况



- 设备开发调优的主要挑战及关键技术
- 镜像分析工具解读
- **栈估算分析工具解读**
- 性能分析工具解读
- 可视化Trace工具解读
- 轻量级内存检测工具解读

栈估算分析工具解读

开发者典型场景

- 运行态内存紧缺，任务栈设置过大浪费资源，过小导致爆栈
- 如何合理设置任务栈大小？



02 栈估算工具，分析优化任务栈大小

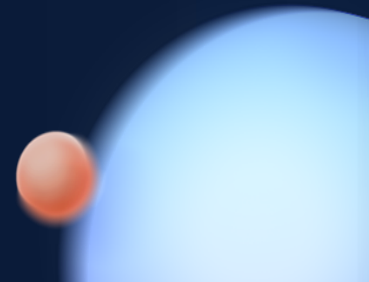
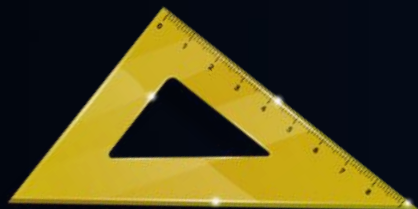
静态栈估算工具通过遍历反汇编文件，计算函数的栈开销与函数调用关系，从而估算任务栈开销，为LiteOS内核当中栈溢出分析、栈空间优化提供数据参考。

工具可提供功能

- 提供函数调用关系图
- 提供函数最大栈开销、函数内部栈开销、所在代码行号

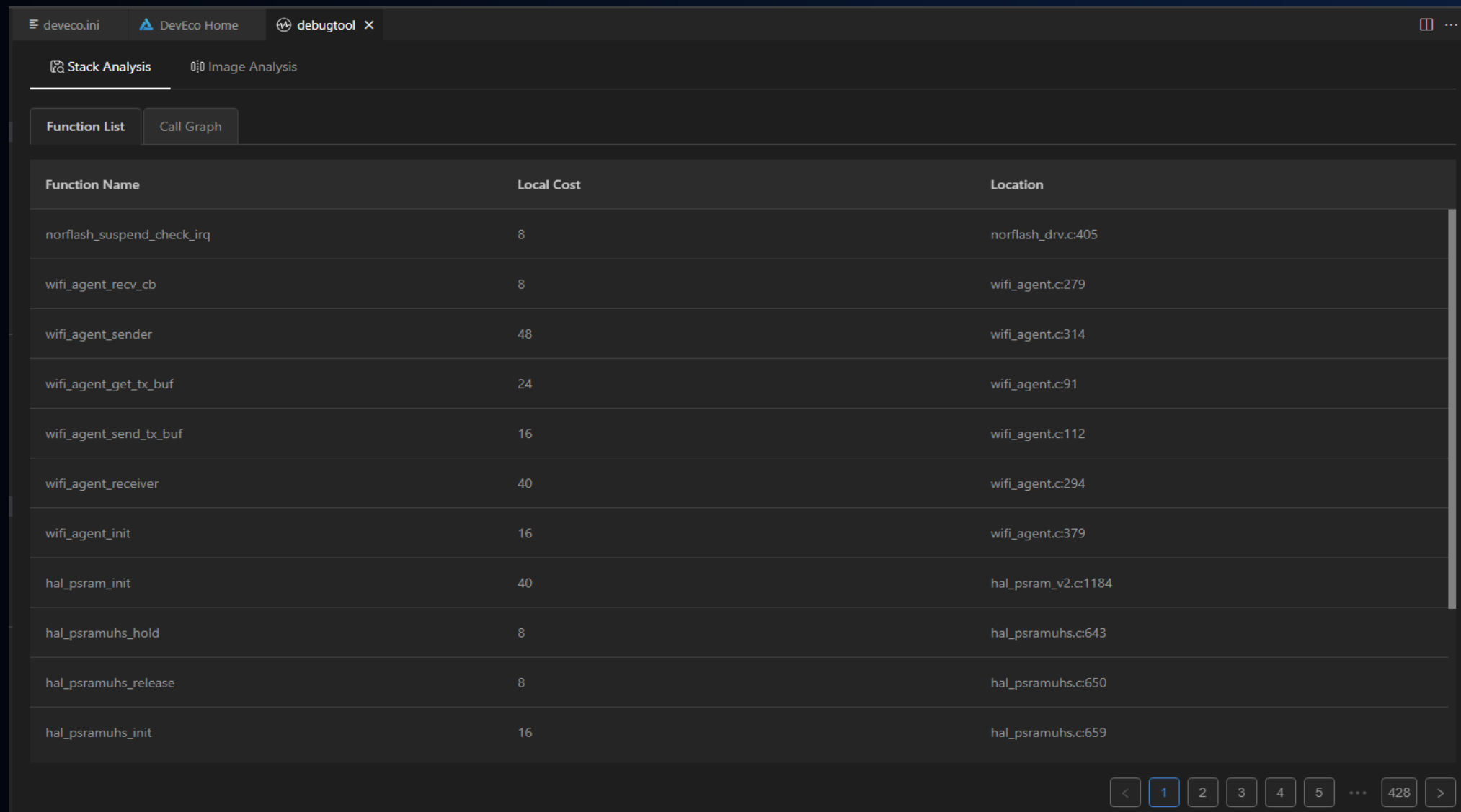
开发者收益

- 减少爆栈风险
- 节省内存，优化栈空间



栈估算分析工具解读

支持查看函数内部栈开销及所在行号



The screenshot displays the 'Stack Analysis' tool interface. At the top, there are tabs for 'Stack Analysis' and 'Image Analysis'. Below the tabs, there are two buttons: 'Function List' (selected) and 'Call Graph'. The main area shows a table with the following columns: 'Function Name', 'Local Cost', and 'Location'. The table lists several functions with their respective costs and source file locations. At the bottom right, there is a pagination control showing page 1 of 428.

Function Name	Local Cost	Location
norflash_suspend_check_irq	8	norflash_drv.c:405
wifi_agent_rcv_cb	8	wifi_agent.c:279
wifi_agent_sender	48	wifi_agent.c:314
wifi_agent_get_tx_buf	24	wifi_agent.c:91
wifi_agent_send_tx_buf	16	wifi_agent.c:112
wifi_agent_receiver	40	wifi_agent.c:294
wifi_agent_init	16	wifi_agent.c:379
hal_psram_init	40	hal_psram_v2.c:1184
hal_psramuhs_hold	8	hal_psramuhs.c:643
hal_psramuhs_release	8	hal_psramuhs.c:650
hal_psramuhs_init	16	hal_psramuhs.c:659

栈估算分析工具解读

支持查看函数调用深度、最大栈开销、函数内部栈开销、代码行号

The screenshot shows the 'Stack Analysis' tool interface. At the top, there are tabs for 'Function List' and 'Call Graph', with 'Function List' selected. Below the tabs is a table with the following columns: 'Function Name', 'Depth', 'Max Cost', 'Local Cost', and 'Location'. The table lists several functions, with the first two having expandable icons (+) next to their names. At the bottom right, there is a pagination control showing a range of pages from 1 to 481, with page 2 currently selected.

Function Name	Depth	Max Cost	Local Cost	Location
+ pmu_pll_div_set	3	100	32	pmu_best2003.c:1095
+ pmu_get_efuse	3	92	24	pmu_best2003.c:1156
pmu_sys_freq_config	0	8	8	pmu_best2003.c:2687
hal_cmu_sys_get_freq	0	0	0	
hal_sys_ms_get	0	0	0	
hal_fast_sys_timer_get	0	0	0	
+ hal_sys_timer_calib_end	1	16	16	hal_timer.c:509
+ hal_sys_timer_calib	1	16	8	hal_timer.c:573
hal_sys_timer_ticks_to_us	0	0	0	
__pmu_dig_init_volt_veneer	0	4	4	
__pmu_dig_set_volt_veneer	0	4	4	

- 设备开发调优的主要挑战及关键技术
- 镜像分析工具解读
- 栈估算分析工具解读
- **性能分析工具解读**
- 可视化Trace工具解读
- 轻量级内存检测工具解读

性能分析工具解读

开发者典型场景

- 系统性能差，CPU占用高，不知道耗时在哪里？
- 如何查看高频执行函数及路径？



03 性能分析工具，洞悉系统瓶颈

Perf性能分析工具，基于事件采样统计的原理，实现对LiteOS内核中**高频函数**、**热点路径**的分析，助力识别性能瓶颈。

支持两种工作模式

计数模式：采集事件发生的次数及执行时间

采样模式：采集上下文如PC、BackTrace等，解析出热点函数与热点路径等信息

支持三种类型的采样

- 硬件PMU采样
- 软件打点采样
- 高精度周期采样us

开发者收益

通过Perf识别性能瓶颈，辅助系统性能优化。

性能分析工具解读

性能分析 x

section id: 0 cpu id: 1 在此输入搜索内容 保存文件

Samples: 461

Children	Self	Symbol
100.00%	0.00%	OsTaskExit
> 100.00%	0.00%	OsTaskEntry
> 98.26%	0.00%	PerfTestTask
> 98.26%	0.00%	test
> 98.26%	0.00%	boo
∨ 93.49%	93.49%	bar
bar		
boo		
test		
PerfTestTask		
OsTaskEntry		
OsTaskExit		
> 4.77%	4.77%	foo
> 1.08%	0.00%	thread_loop
> 1.08%	0.00%	TaskRdrFlush
> 1.08%	1.08%	os_cache_flush



- 设备开发调优的主要挑战及关键技术
- 镜像分析工具解读
- 栈估算分析工具解读
- 性能分析工具解读
- 可视化Trace工具解读
- 轻量级内存检测工具解读

可视化Trace工具解读

开发者典型场景

- 如何了解LiteOS内核运行轨迹?
- 如何对系统运行事件进行轨迹追踪?
- 如何对系统运行过程中资源占用情况实时分析?
- 如何可视化显示系统资源占用情况?



04 可视化Trace, 洞悉程序运行轨迹

Trace 即追踪, 通过采用对LiteOS内核静态代码打桩和缓冲区记录方式, 在桩被执行时, 获取事件发生的上下文、系统任务信息, 并写入到缓冲区, 解决了开发者不易获取系统运行轨迹的问题。

工具可提供功能

以图形界面展示事件、CPU、内存、运行轨迹等信息

开发者收益

- 通过Trace了解系统运行的轨迹, 更好地理解系统。
- 通过Trace分析系统异常前的操作, 定位死机死锁问题。

可视化Trace工具解读

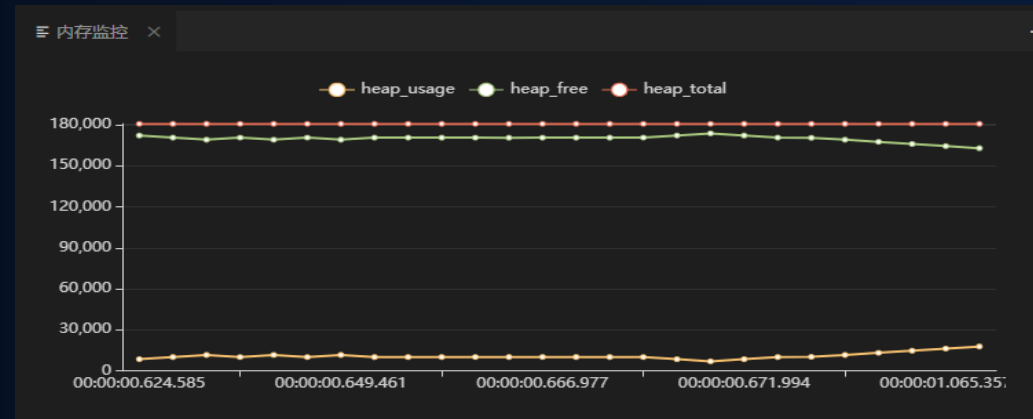
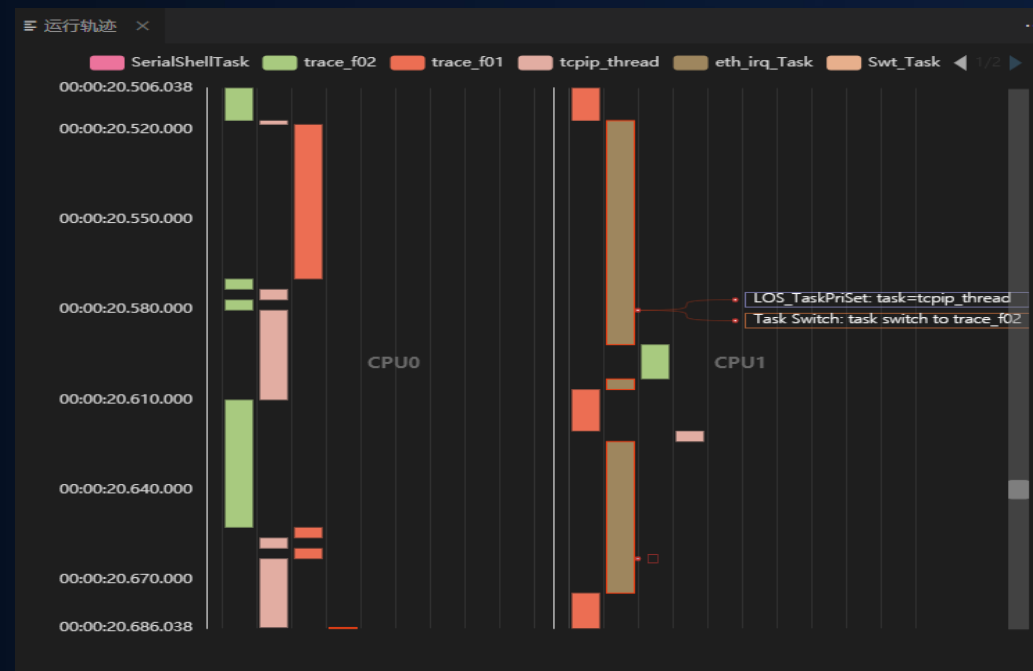
事件信息

请勾选需要查看的事件类型:

Task Queue Semaphore Mutex Timer Hwi Event Heap

[Export to Excel](#)

时间戳	CPU ID	任务名称	事件描述
00:00:20.476.435	0	Interrupt	osInterrupt out: hwiNum=89
00:00:20.488.120	0	Interrupt	osInterrupt in: hwiNum=90
00:00:20.488.148	0	Interrupt	osInterrupt out: hwiNum=90
00:00:20.516.891	0	trace_f02	Task Switch: task switch to tcpip_thread
00:00:20.516.921	1	trace_f01	Task Switch: task switch to trace_f02
00:00:20.518.211	0	tcpip_thread	Task Switch: task switch to trace_f01
00:00:20.567.131	0	Interrupt	Interrupt in: hwiNum=64
00:00:20.569.646	0	trace_f02	Task Switch: task switch to trace_f02
00:00:20.573.120	0	trace_f02	Task Switch: task switch to tcpip_thread
00:00:20.576.611	0	tcpip_thread	Task Switch: task switch to trace_f02
00:00:20.580.065	0	trace_f02	Task Switch: task switch to tcpip_thread
00:00:20.588.038	1	eth_irq_Task	LOS_TaskPriSet: task=tcpip_thread
00:00:20.591.511	1	eth_irq_Task	Task Switch: task switch to trace_f02
00:00:20.599.492	0	tcpip_thread	LOS_TaskPriSet: task=tcpip_thread



- 设备开发调优的主要挑战及关键技术
- 镜像分析工具解读
- 栈估算分析工具解读
- 性能分析工具解读
- 可视化Trace工具解读
- 轻量级内存检测工具解读

轻量级内存检测工具解读

开发者典型场景

- 嵌入式设备资源有限且维测手段有限，软件踩内存问题难定位？
- 结合 pc、lr 等寄存器和asm文件，定位问题速度慢？

```

C lms_demo.c X
demos > lms > C lms_demo.c > LMS_Case_HeapOverflow(VOID)
42 #define TSK_PRIOR_LMS 4
43
44 static UINT32 LMS_Case_HeapOverflow(VOID)
45 {
46     CHAR *buf = NULL;
47     CHAR tmp;
48     UINT32 ret;
49
50     buf = LOS_MemAlloc(m_aucSysMem0, 4); /* mem size 4 */
51
52     tmp = buf[4];
53     tmp = buf[3]; /* Array 3. No exception information is printed on the LMS. */
54     /* Array 4. When the fourth byte overflows, the LMS prints the read error information. */
55
56     printf("buf is %c.\n", tmp);
57     ret = LOS_MemFree(m_aucSysMem0, buf);
58
59     return ret;
60 }
61
62 static UINT32 Example_TaskLMS(VOID)

```

问题 1 输出 终端 调试控制台 串口终端

接收区(字符显示)	端口	COM75	波特率	115200	校验位	None	数据位	8	停止位	1	流控	None
27	[0x20004220]:	88	21	00	20	08	42	00	20			
28	[0x20004228]:	00	06	00	00	0a	0a	0a	0a			
29	[ERR] Accessable heap addr	00										
30	[ERR] Heap red zone	11										
31	[ERR] Heap freed buffer	01										
32												
33	----- LMS BackTrace Start -----											
34	runTask->taskName = LMS_NAME											
35	runTask->taskId = 3											
36	*****backtrace begin*****											
37	traceback 1 -- lr = 0x0801182a -- fp = 0x080117f8											
38	traceback 2 -- lr = 0x080021f6 -- Function Name: LMS_Case_HeapOverflow											
39	traceback 3 -- lr = 0x08010f24 -- File: d:\10_LiteOSOpenSource\Codehub-LiteOS\Huawei_LiteOS\demos\lms\lms_demo.c											
40	traceback 4 -- lr = 0x08010f11											
41	traceback 5 -- lr = 0x08019d4a -- fp = 0x08010f98											
42	traceback 6 -- lr = 0x08019e44 -- fp = 0x08019db8											
43	traceback 7 -- lr = 0x08006486 -- fp = 0x08019e1c											
44	buf is L.											
45	TaskLms LOS_TaskDelay Done.											
46												

05: 轻量级内存检测，一键定位内存问题

轻量级内存检测，通过使用影子内存映射标记系统内存状态，可快速解决LiteOS内核中内存越界访问、释放后访问、多重释放等内存疑难杂症，支持如下功能：

工具可提供功能

- 实时检测多种内存问题，如：
缓冲区溢出、释放后使用、多重释放和释放野指针等
- 检测到错误时实时打印回溯栈

开发者收益

结合backtrace，快速一键式定位问题代码段。



谢谢